

ISAW

1.3.0

ARGONNE NATIONAL LABORATORY

---

Intense Pulsed Neutron Source

# ISAW User Manual

INTENSE PULSED NEUTRON SOURCE

# ISAW User Manual

---

Dennis Mikkelson, DataSet Tools, Netcomm packages  
Ruth Mikkelson, Command package NeXus data retriever  
Peter Peterson, Linux install, SCD operators  
Alok Chatterjee, ISAW GUI package  
John Hammonds, IPNS package  
Chun Loong, Scientific Advisor  
Dongfeng Chen, ChopTools package  
Kevin Neff, initial OverPlotView package  
Sumathi Thiagarajan, initial NeXus data retriever  
Thomas Worlton, project manager

The research is sponsored by the U.S. Department of Energy under contract no. W-31-109-ENG-38.

©Argonne National Laboratory  
9700 South Cass Ave • Building 360  
Phone 630-252-8755 • Fax 630-252-4163

---

# Table of Contents

<b>TABLE OF CONTENTS</b> .....	<b>1</b>
<b>TABLE OF FIGURES</b> .....	<b>3</b>
<b>INTRODUCTION</b> .....	<b>1</b>
THE ISAW GRAPHICAL USER INTERFACE .....	1
<i>The ISAW menu bar</i> .....	1
<i>The tree view</i> .....	2
<i>Tabbed Panes</i> .....	2
The Attributes tab .....	2
DataSet Log tab .....	2
Session Log tab.....	2
System Properties tab .....	2
Live Data connection tab.....	2
Scripts tab .....	2
<b>INSTALLING ISAW</b> .....	<b>5</b>
STEPS REQUIRED FOR ISAW INSTALLATION .....	6
<i>Install Java</i> .....	6
<i>Copying the ISAW distribution file(s)</i> .....	7
<i>Unzipping the ISAW distribution</i> .....	7
<i>Creating shortcut commands to start the application</i> .....	7
CUSTOMIZING ISAWPROPS.DAT .....	8
<i>Customizing default local file locations</i> .....	8
<i>Specifying Live Data Server connections</i> .....	8
<i>Specifying Remote Data Server connections</i> .....	9
<i>Customizing Viewer Defaults</i> .....	9
Supported ColorScale names: .....	9
Possible RebinFlag values:.....	10
Possible HScrollFlag values:.....	10
Valid ViewAzimuthAngle values: .....	10
Valid ViewAltitudeAngle values: .....	10
Valid ViewDistance values: .....	10
Possible values for the ViewGroups flag .....	10
Possible values for the ViewDetectors flag .....	10
Valid Brightness values:.....	10
Valid Auto-Scale values:.....	10
<b>ISAW DATA MODEL</b> .....	<b>11</b>
DATA OBJECTS .....	11
DATASET CREATION .....	11
DATASET ORGANIZATION .....	11
DATASET OPERATORS .....	12
<b>READING/SAVING DATA</b> .....	<b>13</b>
THE IPNS RUNFILE RETRIEVER .....	13
THE NEXUS FILE RETRIEVER .....	13
LOG FILE RETRIEVER .....	14
LIVE DATA RETRIEVER.....	14
REMOTE FILE SERVERS .....	16
SAVING DATA .....	17
<b>DATA VIEWERS</b> .....	<b>18</b>
<i>Viewer menus</i> .....	18
THE IMAGE VIEWER.....	18
<i>Line graphs</i> .....	19
<i>Controls</i> .....	19
<i>Cursor Readouts</i> .....	20
<i>Sort order display</i> .....	20
<i>Selected spectrum marker</i> .....	20
<i>Zoom control</i> .....	20
<i>Auto-scale</i> .....	20
THE SCROLLED GRAPH VIEWER.....	20
THE SELECTED GRAPH VIEWER.....	21
THE 3D VIEWER.....	21
THE TABLE VIEWER.....	21
<i>Generating a table</i> .....	21
<i>Copying data from a table to another application on a PC</i> .....	22
<b>COMMAND SCRIPTS</b> .....	<b>23</b>
THE GUI ELEMENTS .....	23
<i>The Editor Pane</i> .....	23
<i>The Immediate Pane</i> .....	23
<i>The Buttons</i> .....	23
<i>The Status Line</i> .....	23
DATA TYPES .....	23
OTHER DATA TYPES: .....	24
NUMERIC, LOGICAL, AND RELATIONAL OPERATIONS .....	24
STRUCTURES .....	25
<i>For Structure:</i> .....	25
<i>If Structures:</i> .....	25
Examples: .....	25
<i>On Error Structures:</i> .....	25
INTRINSIC OPERATORS .....	26
<i>Load</i> .....	26
<i>Display</i> .....	26
<i>Send</i> .....	26
<i>Save</i> .....	26
<i>Return</i> .....	27
OTHER OPERATORS .....	27
<i>Java-class operators</i> .....	27
<i>Script Operators</i> .....	27
WRITING JAVA OPERATORS .....	28
<i>Input-Output Considerations:</i> .....	28
<i>[Input-]Output Parameters:</i> .....	28
<i>Return value:</i> .....	28
<i>Considerations for other cases:</i> .....	28
INTERFACE TO ISAW .....	28
<i>Isaw to CommandPane:</i> .....	28
<i>Isaw to ScriptProcessor:</i> .....	29
<i>CommandPane to Isaw:</i> .....	29
<b>CHOPPER ANALYSIS PACKAGE</b> .....	<b>30</b>
HISTORY .....	30
DATASET EVALUATION .....	31
COMBINING MULTIPLE RUNS AND CONVERSION TO VARIOUS SCATTERING FUNCTIONS .....	32
FUTURE PLANS.....	36
<b>BUILDING DATASETS</b> .....	<b>37</b>
<i>Major steps in building a DataSet:</i> .....	37
CREATING A DATA SET: .....	37
<i>Construct the empty DataSet</i> .....	37
<i>Add "attributes" to the DataSet</i> .....	38
<i>Construct a Data object</i> .....	39
<i>Add attributes to Data object</i> .....	40
<i>Add the Data object to the DataSet</i> .....	40
ATTRIBUTES NEEDED IN A DATA SET AND DATA OBJECT: .....	40
A DATA RETRIEVER: .....	42
AN EXAMPLE: .....	42
<b>REFERENCES</b> .....	<b>46</b>

**TABLE OF CONTENTS**

INDEX .....47

## TABLE OF CONTENTS

### Table of Figures

**Error! No table of figures entries found.**

Figure 4- 1. A file selection dialog box generated by the ISAW “Load Runfile(s)” menu item selection. ....	14
Figure 4- 2. The communication paths and protocols for live data access. ....	15
Figure 4- 3. The menu paths and associated display for live data server connection. ....	15
Figure 4- 4. The live data display dialog box. DataSet 0 is a monitor DataSet. Checking the “Show” box on the second DataSet causes ISAW to read the remote data and pop up a view of the DataSet. Checking the “Auto” box would cause ISAW to read the remote data at at the interval selected by the slider at the bottom. ....	16
Figure 5- 1. An ISAW image view of time-of-flight neutron scattering data. ....	19
Figure 5- 2. A scrolled graph view of a sample DataSet. The graphs can be scrolled vertically and the cursor can read out the data at any position on any of the graphs. ....	21
Figure 5- 3. A 3D view of a DataSet. The red line marks the incoming beam direction. The green line marks the axis at 90 degrees to the incident beam in the scattering plane. The blue line indicates the vertical axis. The buttons allow you to either step through the time channels or to scroll steadily through the time channels. The cursor can be used to select a detector element whose attributes will be displayed. ....	22

---

## Introduction

*Integrated Spectral Analysis Workbench software (ISAW) provides tools for reading, manipulating, and visualizing neutron scattering data.*

### The ISAW Graphical User Interface

ISAW is a java based software package that provides a foundation for reading, manipulating and visualizing data [1]. A Graphical User Interface (GUI) enables intuitive operation of the software package [2]. The GUI is divided into two main panels plus a menu bar at the top and a message area at the bottom. The relative sizes of the two areas may be adjusted by dragging the gray stippled divider bars or by clicking on one of the triangles in the bar. The entire ISAW window can be resized by dragging the corner of the window.

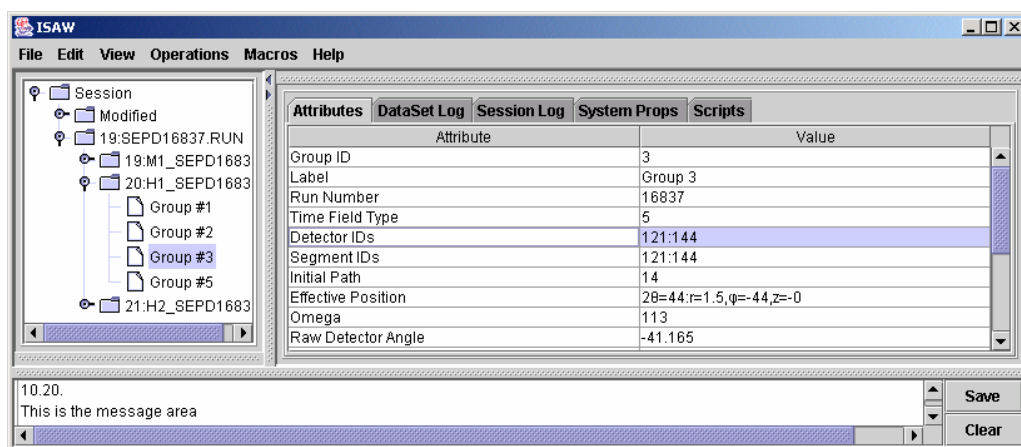


Figure 1- 1 The ISAW graphical user interface (GUI).

### The ISAW menu bar

The ISAW menu bar is used to load files, create views of the data, perform operations on the data, and to select options. Loading and saving of data is initiated through the **File menu**. The **Edit menu** allows you to remove nodes from the tree and/or modify DataSet attributes. The **View menu** allows you to create DataSet viewers. The viewers themselves also include a menu bar that can be used to change the type of view they show and the options for that view.

## INTRODUCTION

The **Operations menu** is context sensitive. The operations listed are the ones available for the object selected. If no DataSet is selected the operations menu may be empty. The **Macro menu** contains macros or scripts developed for each instrument. The **Help menu** provides information about using ISAW.

### The tree view

The left panel of the GUI shows a tree view of loaded files, spectra (Data objects) and sets of spectra (DataSet object). When you create new datasets through modifying another DataSet, the new tree node goes under the “Modified” node. DataSets and containers for DataSets are displayed as folder icons in the tree. Spectra or group data are shown as document icons in the tree.

### Tabbed Panes

The right panel includes a number of tabbed panes used to display log information, detector information, computer system properties, and a command pane for editing and executing scripts.

#### *The Attributes tab*

The bottom left panel shows the attributes of the selected DataSet or Data block. Attributes may be numbers or text data. One attribute, the detector position, contains a triad of numbers, which vary according to the chosen coordinate system. This pane is updated dynamically when different spectra or DataSets are selected.

#### *DataSet Log tab*

When a DataSet is selected in the tree view, the DataSet Log pane shows the log of actions taken on the selected DataSet.

#### *Session Log tab*

One of the tabbed panes shows all commands issued during the current session of running ISAW.

#### *System Properties tab*

The System Properties tabbed pane provides a list of the properties of the system when running ISAW. These properties include the ISAW build date and time, the amount of memory in use by the Java virtual machine (JVM), the Java version, and the paths to some of the software components.

#### *Live Data connection tab*

When viewing live data, an additional tabbed pane will appear to show data sets and controls available from the chosen live data server. There is an “Exit” button to terminate live data connections.

#### *Scripts tab*

Although a GUI is excellent for ease of use for simple operations, it becomes tedious if operations must be repeated for multiple sets of data or a set of operations must be repeated many times. To allow repetitive operations, we have included a command language and a command interpreter. The language is similar to Fortran or Basic and is

## INTRODUCTION

quite easy to learn. Scripts can also be reached through the “Load Script” item on the File menu or through the “Macros” menubar item

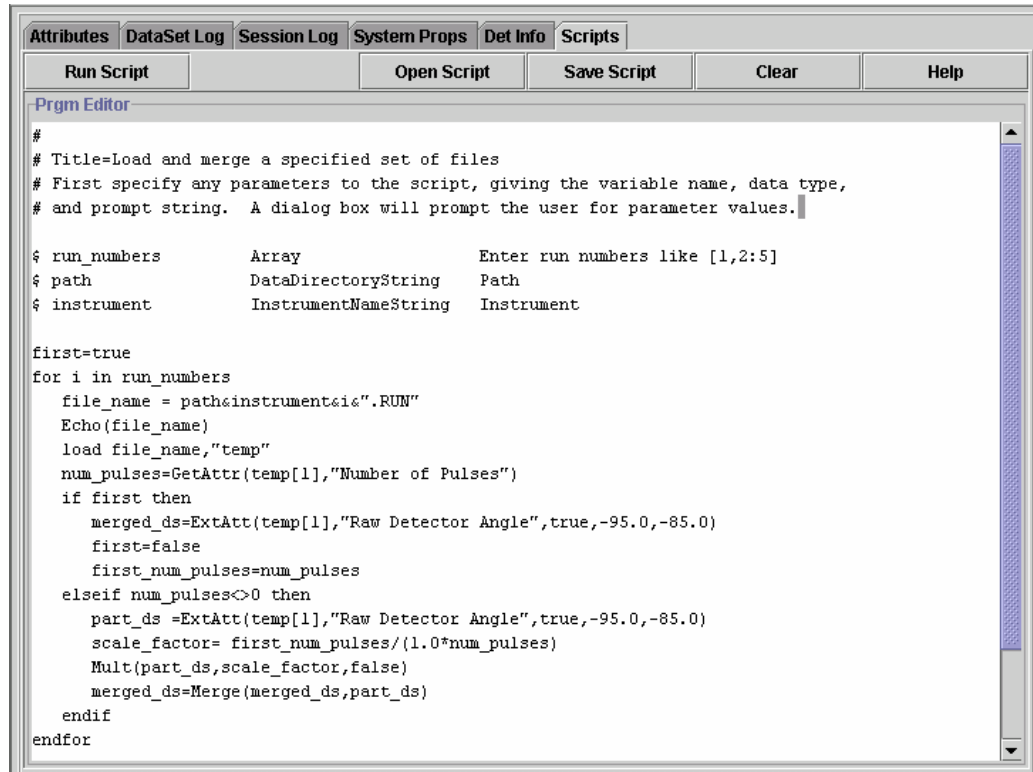


Figure 1- 2. A sample ISAW script.

Scripts can pop up dialog boxes to ask for parameters needed by the script and can load file data and send data to the tree for further processing through GUI interaction. The command "\$ Title =...", gives a title to the dialog box that will be created. The other four lines beginning with the "\$" signal the command interpreter that four parameters must be input by the user. The first field in these records is the parameter name, the second is the parameter data type and the third field is the prompt to use on the dialog box presented to the user. DataDirectoryString and InstrumentNameString are special data types that get their default values from parameters in IsawProps.dat. The script in Fig. 1.2 will pop up the dialog box shown in Fig. 1.3.



## INTRODUCTION

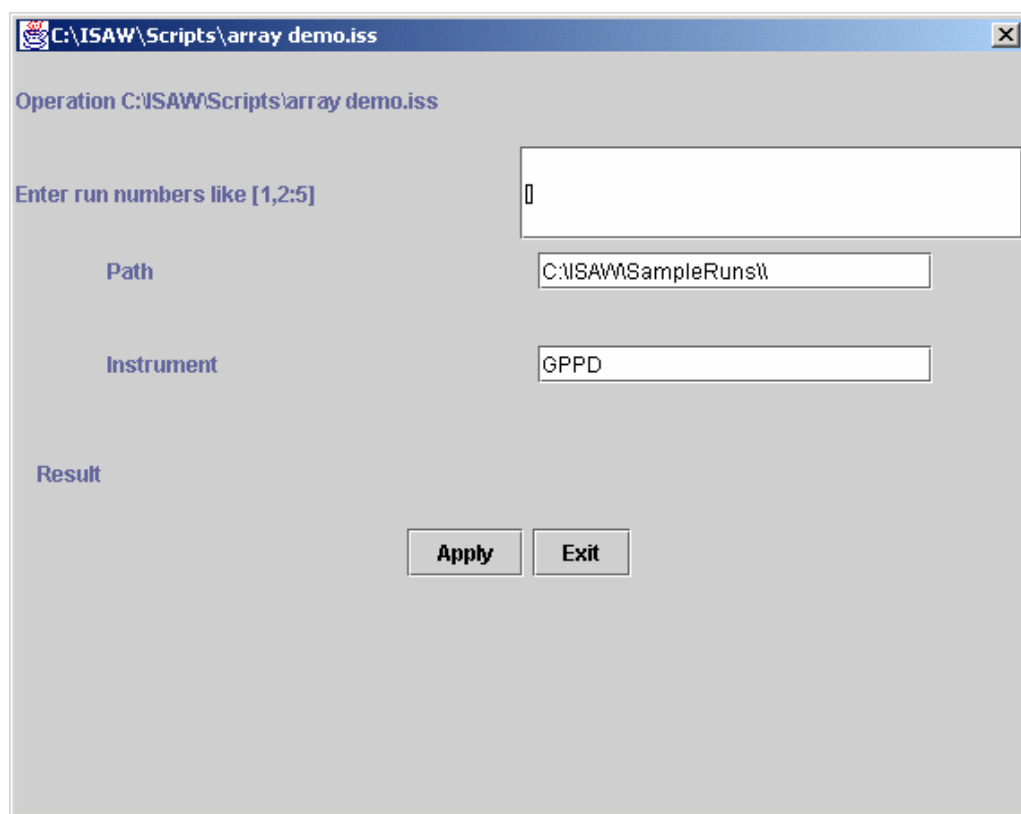


Figure 1- 3. The dialog box generated by the ISAW script in Figure 1-2.

The default values of the “Path” and “Instrument” come from the IsawProps.dat file, but may be changed by the user. When a script specifies a DataDirectoryString variable as a parameter type, ISAW gets the value of “Data\_Directory” from IsawProps.dat in your home directory. When a script specifies an InstrumentNameString data type as a parameter, ISAW gets the value of “Default\_Instrument” from IsawProps.dat as the default value.

There are a number of sample scripts included in the scripts folder of the ISAW distribution. There are additional sample data files on the ISAW ftp site in the folder “MoreSampleData”. The series of GPPD run files from 11471 to 11603 can be used with the sample script “array demo.iss” or “merge\_consecutive\_runs.iss” to show how a set of runs can be merged, operated on, and displayed.

Detailed discussion of ISAW Installation, the ISAW data model, reading/saving data, data viewers, command scripts, and the chopper analysis package[3] is done in subsequent sections of this manual.

## Installing ISAW

*ISAW software is freely available and easy to install, but requires java, HDF, and NeXus libraries. The “Selected Graph View” also uses the SGT\_V2 library. These libraries are also free.*

One advantage of writing software in the Java language is portability. Files containing Java language source code have the extension “. java”. The compiled Java byte code files have extension “.class”. These class files can be created on one computing platform and run on any computing platform with at least as high a version of java .exe. Developing java programs and compiling them requires the java development kit, which can be obtained for free from Sun’s web site, <http://java.sun.com/j2se/>. If you just want to run a java program someone else has developed, you only need the java runtime environment, which can be obtained from the Java web site.

Our initial plan with ISAW was to write all the code in java. This was possible until it came time to add support for reading of NeXus files. The NeXus API (see <http://www.neutron.anl.gov/nexus/>) is layered on top of the HDF library developed at the National Center for Supercomputer Applications (see <http://hdf.ncsa.uiuc.edu/>). These libraries are written in C or C++, so require separate versions for different computer systems. We have included the essential parts of Java, NeXus, and HDF for distribution from the ISAW web site. If you already have these components installed, all that is necessary to install ISAW is to copy the ISAW files (see <ftp://zuul.pns.anl.gov/isaw/>) to a folder on your system. All ISAW files can be removed by deleting the ISAW folder and deleting the properties file `IsawProps.dat` from your home directory.

There are a number of ways to run Java applications. From the standpoint of ease of distribution and use, the simplest is to place all the java class files in a java archive or “jar” file.

A jar file is actually a zip archive and can be handled with standard zip utilities. The main difference between a jar file and a zip file is that the jar file contains a “manifest”

---

## INSTALLING ISAW

which can be used to tell Java which is the “main” class to run when you are running an application from a jar file.

All ISAW class files are contained in “isaw.jar” and ISAW can be run without extracting the files from the archive. It is also possible to “unjar” the files into a regular directory structure and to run the application using the unjarred files. This is useful when you are doing development and want to edit and compile individual files, but for distribution it is much easier to distribute a single archive file.

As mentioned at the beginning, in order to provide support for NeXus files, it is necessary to include the HDF and NeXus libraries. The Scientific Graphics Toolkit, sgt\_v2, is a general purpose Java graphics package developed at the National Oceanographic and Atmospheric Administration (NOAA). It is only used in the “Selected Graph View” and could be omitted if that view is not desired. The SGT package is in folder “gov” and is bigger than the entire ISAW package. More information about SGT can be obtained from <http://www.epic.noaa.gov/java/sgt/index.html>. Other files included besides the ISAW class files and these libraries are sample scripts, sample data, Help, and Documentation. To further simplify distribution, these files and libraries are zipped or archived together. A single install jar file can be used on Windows, Linux, or UNIX systems.

### Steps required for ISAW installation

The first time ISAW is installed it is necessary to ensure that the following steps are completed.

- **Install java** on the computer to be used for ISAW
- **Copy the ISAW distribution file(s)** to the local computer
- **Unzip the ISAW files** to a folder on the local disk
- **Create shortcut commands** to start the application
- **Customize the default file locations** in IsawProps.dat

### Install Java

Before ISAW can be installed or run, it is necessary to install the Java Run time Environment (jre) or the Java Distribution Kit (jdk) on your computer. We currently recommend Java version 1.3.1\_03, although Java 1.4 is available. We recommend using the jre instead of the jdk unless you are going to do software development. For software and instructions, see

<http://java.sun.com/j2se/1.3/>

After installation, the Java executable, `java.exe`, will be in the “bin” folder of the main `jre` or `jdk` folder. The command to start ISAW requires that either the path to this command is included on the command line or that the path environmental variable includes that folder. You can test whether your path environmental variable includes `java.exe` by opening a command pane and typing the command “`java -version`”. If Java is correctly installed, this will tell you which version of Java is installed. If you get an error message, it will be necessary to either modify your path environmental variable to include the Java “bin” folder in the path or to use an explicit path when typing the `java` command.



On a Windows 2000 computer, the path can be modified through the system icon of the control panel. You can start the control panel by selecting `start→settings→control panel`. You would then double-click on the system icon and select the “Advanced” tab. Then push the “Environmental Variables” button. There should be a “Path” entry in the “User variables” section. Edit this entry to include the path to the “bin” folder of your `jre` or `jdk` software. Changes do not take effect until you create a new command window. For more details and instructions on other windows versions, see <http://www.javasoft.com/j2se/1.3/install-windows.html>.

### Copying the ISAW distribution file(s)

The ISAW files are available for distribution through the ftp site <ftp://zuul.pns.anl.gov/isaw>. The ISAW class files in “`isaw.jar`” are identical for different platforms, but installation procedures and run procedures are slightly different. The non-ISAW files are different for different platforms. The HDF and NeXus libraries used for reading NeXus files contain native methods (code produced by C and C++ compilers). All the different files have been packed inside an “install” executable, so only one file needs to be downloaded. The name of the installation file as of this revision of the manual was “`Isaw-130-install.jar`”.

### Unzipping the ISAW distribution



The ISAW distribution is a self-extracting Java archive (`jar` file). To extract the files and install them on your computer, double-click the downloaded file, and then choose the destination directory for the files. We recommend `C:\ISAW`, but you may put them at any desired location. Make a note of the location since ISAW will need to be run from that folder.

### Creating shortcut commands to start the application



For Windows systems we have created the file “`PC_run.bat`” to run the application. This file adds the folder containing the HDF and NeXus libraries to the path and runs ISAW. A shortcut to the bat file can be created by right clicking on the file icon and dragging it to the desktop, then selecting “Create shortcut here”. We recommend renaming the shortcut to “ISAW”. If you want to be able to start ISAW from the Start Menu, you can drag and drop the shortcut to the Start button. If you

## INSTALLING ISAW

later decide to remove ISAW, you will need to right-click on the Start button and open the start menu so you can delete the ISAW menu item.

### Customizing IsawProps.dat

ISAW makes use of a text file, IsawProps.dat, in your home directory to determine default values of many parameters. If you already have this file in your home directory, ISAW will use it. If you do not have this file, it will be created the first time you run ISAW, but you will probably want to modify it to suit your purposes.

To edit properties, select “Edit Properties” from the **File Menu** in ISAW. “IsawProps.dat” is a text file, so it can also be modified using any editor that works with text files. If you change the location of ISAW files, you will need to modify “IsawProps.dat” to specify the new location of ISAW files.

### Customizing default local file locations

IsawProps.dat” specifies default locations for files as well as Internet addresses for nodes running live data servers or file data servers that can be accessed by ISAW. You may want to edit “IsawProps.dat” to change paths specifying data and/or script locations. You may also want to reference other instruments and/or live data servers by changing this file. The following entries in IsawProps.dat on a Windows PC define the default local file locations.

```
ISAW_HOME=C:/ISAW/  
Help_Directory=C:/ISAW/IsawHelp/  
Script_Path=C:/ISAW/Scripts/  
Data_Directory=C:/ISAW/SampleRuns/  
Instrument_Macro_Path=C:/ISAW/  
User_Macro_Path=C:/ISAW/  
Image_Path=C:/ISAW/images/
```

### Specifying Live Data Server connections

ISAW can connect to live data servers that can provide data as it is collected. Entries specifying live data servers would be of the form:

```
Inst1_Name=HRMECS  
Inst1_Path=zeus.pns.anl.gov;6088  
  
Inst2_Name=mandrake  
Inst2_Path=mandrake.pns.anl.gov;6088  
  
Inst3_Name=GPPD  
Inst3_Path=gppd-pc.pns.anl.gov;6088  
  
Inst4_Name=UW-Stout  
Inst4_Path=dmikk.mscs.uwstout.edu;6088
```

## INSTALLING ISAW

The number following the semicolon is the tcp port number used to talk to the live data server. The default port number is 6088, but the live data server can be given a different port to use when it is started using the “-T <port>” option. The instrument names will appear in the live data server menu and are just used as a way to link to the path, so they may be modified as desired.

### Specifying Remote Data Server connections

ISAW includes the ability to retrieve datasets from remote data servers. The menu of remote data servers and the paths to those data servers are specified in IsawProps.dat. Entries specifying remote file servers would be of the form:

```
IsawFileServer1_Name=HRMECS(zeus)
IsawFileServer1_Path=zeus.pns.anl.gov;6089

IsawFileServer2_Name=Test(dmikk-Isaw)
IsawFileServer2_Path=dmikk.mscs.uwstout.edu;6089

NDSFileServer1_Name=Test(dmikk-NDS)
NDSFileServer1_Path=dmikk.mscs.uwstout.edu;6008
```

The default port for the file data servers is 6089, but this can be changed when the server is started. The default port for the NDS file server is 6008.

### Customizing Viewer Defaults

The IsawProps.dat file contains default values for many viewer settings as well as parameters mentioned above. The color scale, 3D view position, horizontal scrolling flag, rebinning flag, brightness, horizontal graph scale factor and whether 3D viewers show groups or detectors by default can all be controlled. A sample set of values for viewer defaults in IsawProps.dat is shown below:

```
ColorScale           = RaINbow
RebinFlag            = faLSe
HScrollFlag          = true
ViewAltitudeAngle    = 10
ViewAzimuthAngle     = 50
ViewDistance         = 15
ViewGroups           = MeDium
ViewDetectors        = NOT DRAWN
Brightness           = 40
Auto-Scale           = 0
```

As these examples indicate, the String values are NOT case sensitive. The possible values for viewer options are listed below:

*Supported ColorScale names:*

- > Gray
- > Negative Gray
- > Green-Yellow

## INSTALLING ISAW

- > Heat 1 (default)
- > Heat 2
- > Rainbow
- > Optimal
- > Multi
- > Spectrum

### *Possible RebinFlag values:*

- > true (default)
- > false

### *Possible HScrollFlag values:*

- > true
- > false (default)

### *Valid ViewAzimuthAngle values:*

- > -180 to 180

### *Valid ViewAltitudeAngle values:*

- > -89 to -89

### *Valid ViewDistance values:*

- > positive float values, will be clamped to  $[r/2, 5r]$
- > where  $r$  is the maximum sample to detector distance

### *Possible values for the ViewGroups flag*

- > Small
- > Medium
- > Large
- > NOT DRAWN

### *Possible values for the ViewDetectors flag*

- > Filled
- > Hollow
- > Marker
- > NOT DRAWN

### *Valid Brightness values:*

- > 0 to 100

### *Valid Auto-Scale values:*

- > 0 to 100

## ISAW Data Model

*ISAW uses the Model View Controller architecture. The “model” is a DataSet that groups one or more Data blocks with the same units.*

ISAW viewers and operators work on DataSet objects. DataSet objects have attributes and may contain one or more Data objects (spectra).

### Data objects

The individual spectra are held in "Data" objects. Each Data object contains a spectrum together with errors and a list of attributes associated with the spectrum, such as the effective detector position, detector IDs, etc. These spectra Data objects are contained in "DataSet" objects. In addition to the spectra, a DataSet object contains a list of attributes associated with the entire set of spectra, a list of operator objects that can be applied to the DataSet and a log of operations that have been carried out on the DataSet.

### DataSet creation

Some basic operations, such as addition and scalar multiplication, are supported by all DataSets. However, DataSets come from several different classes of instruments and some more complex operations are not applicable to all types of instruments. Each DataSet maintains a list of relevant operators that can be applied to it. The list of appropriate operators is assigned to a DataSet when it is created by a DataSetFactory object.

### DataSet organization

An IPNS Run file generally contains data taken on a single sample at constant conditions. It is often useful to combine the data from measurements taken under different conditions. This can be done by merging DataSets from different run files. It can also be done by combining DataSets into a single experiment. Currently there are no operations that can be done on collections of DataSets, but it is useful to be able to store all the DataSets comprising an experiment in one file. To combine DataSets, select the DataSet or DataSets and right click on them. You can then elect to “send to” a common container. The “Send to” menu options are shown in the figure.

---



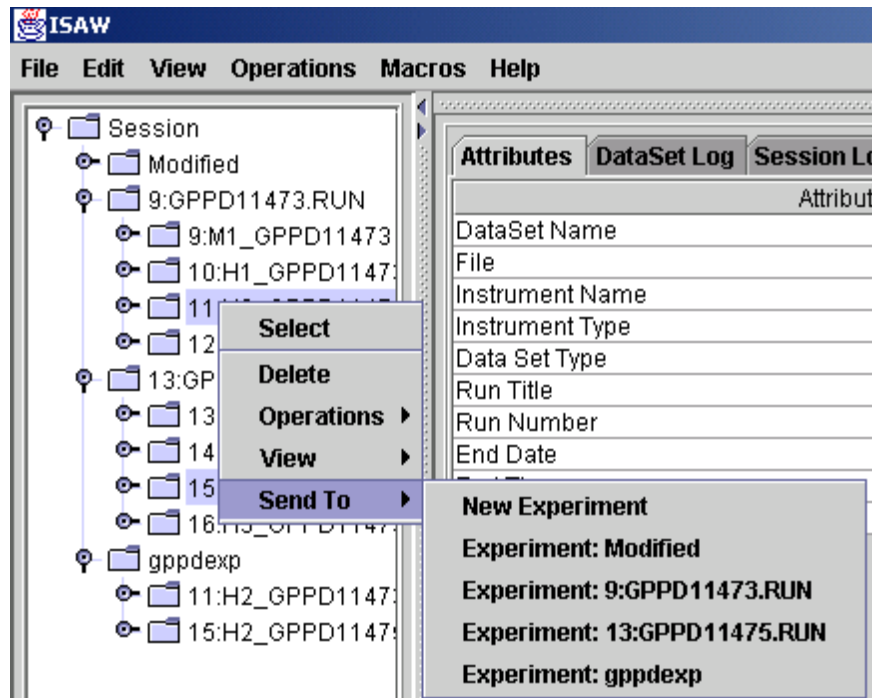


Figure 1. The pop up menu produced by right-clicking on selected DataSets.

### DataSet Operators

Each operator is an object that includes a list of parameters that the user can specify through a generic dialog box. When the user pulls down the **Operations menu** from the menu bar the menu is automatically populated with the names of the operator objects included in the current DataSet. When the user selects a particular operator, a generic dialog box pops up that allows the user to specify the parameters needed by the operation. The dialog box obtains the list of required parameters from the operator object and displays appropriate GUI components based on the types of parameters that are to be specified. This generalization of operators makes it very simple to create new operations and no changes to the user interface are needed when a programmer adds a new operation. The name of the new operation appears automatically on the operations menu and an appropriate parameter input dialog box is generated automatically.

## Reading/Saving Data

*ISAW currently supports reading IPNS Run files and NeXus files. It also supports reading data sent from a Live Data Server or a File Data server.*

ISAW is designed in a modular fashion with data input functions performed by data retrievers.

### **The IPNS Runfile Retriever**

The IPNS run file retriever is used to read IPNS binary data files. The Java code for this retriever is included in a separate package, IPNS. These routines would not be needed for reading NeXus files. From the File menu of the ISAW menu bar you can choose to “Load Data file(s)”. This will bring up a file dialog box with a default location determined by the “Data\_Directory” path specified in IsawProps.dat. When you choose which files to load, ISAW will call the appropriate data retriever to load the data files and list them as nodes in the tree view. ISAW may also automatically produce an image view of the DataSet corresponding to the first histogram loaded.

### **The NeXus File Retriever**

The NeXus file retriever is activated through the same menu choice as the IPNS Runfile retriever. ISAW will determine the file type and call the appropriate retriever to read the file. If the NeXus file contains detector position information stored in the way understood by ISAW, you will be able to operate on the data as well as visualize it.

---

## READING DATA

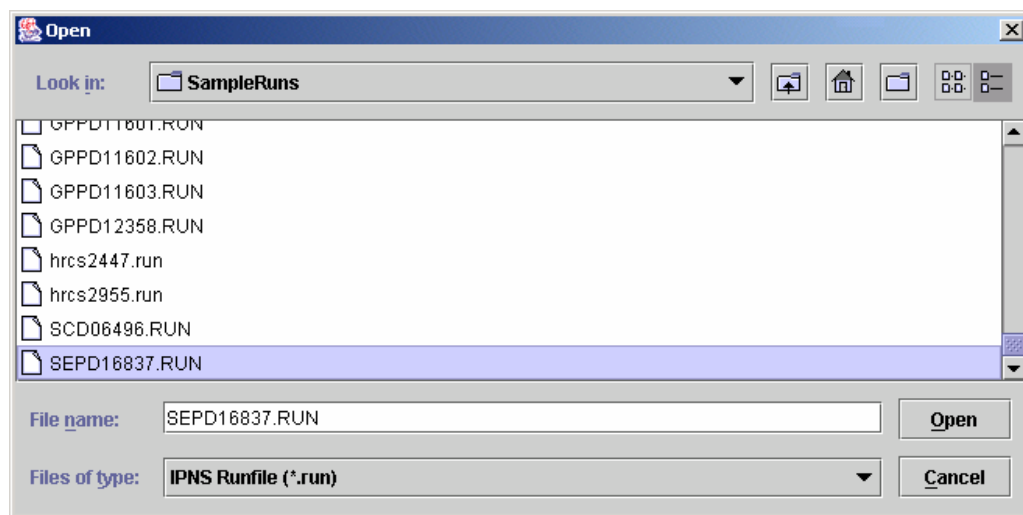


Figure 4- 1. A file selection dialog box generated by the ISAW “Load Runfile(s)” menu item selection.

### Log File Retriever

Log files are often accumulated in conjunction with data collection. At IPNS, we have decided to use the APS Self Describing Data Set (sdds) log files (see <http://www.aps.anl.gov/asd/oag/manuals/SDDStoolkit/SDDStoolkit.html> ). The log files can be read through the same dialog as run files except that you must use the down arrow to select files of type “SDDS Files”. The log file retriever will split up the different log datasets read according to the units of the variable saved. The one of most interest will probably be “SampleTemperature”.

### Live Data Retriever

One of the prime reasons for using Java for this application was to allow easy integration with the Internet and the World Wide Web. Java includes built-in tools to allow interaction with network objects on different nodes on the Internet. Viewing of live data in ISAW involves two other components. A “data sender” runs on the data acquisition system, and a “data server” runs on the control computer used to set up and control data collection for that instrument.

## READING DATA

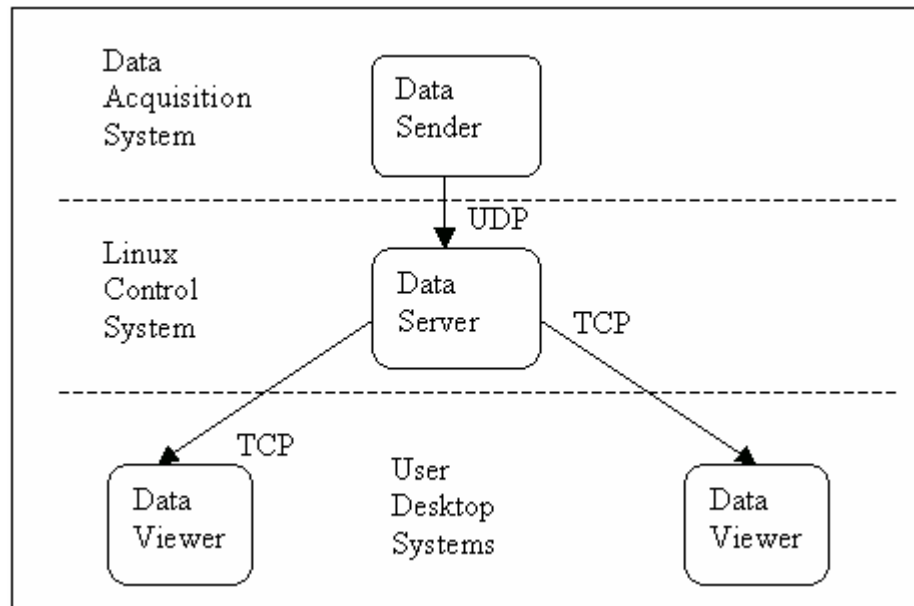


Figure 4- 2. The communication paths and protocols for live data access.

The data sender uses the UDP protocol to send spectra to the data server. The data server accepts packets from one or more data senders and sends spectra to ISAW clients as requested by ISAW. The data server talks to data senders using UDP protocol and talks to ISAW clients using TCP protocol.

Connection to a live data server is initiated through the “Live” item on the “Load Data” sub menu of the File menu. This displays a list of the live data servers from IsawProps.dat and, when the cursor is over the server name, also shows the Internet node name of the node containing that data, as shown in the Figure.

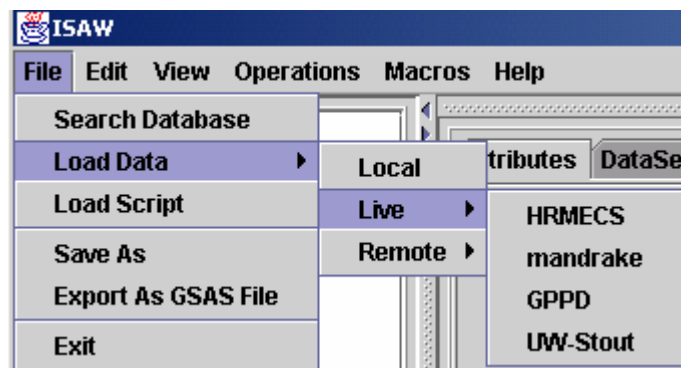


Figure 4- 3. The menu paths and associated display for live data server connection.

The only information this menu provides to the live data code is the node name. The Internet node name of the selected node is used to inquire what DataSets are available on that node. If a connection is successful, a tabbed pane listing the data source node

## READING DATA

name and the live DataSets is created. The user is presented with options “**Show**”, “**Update**”, “**Auto**”, and “**Record**”. The “**Show**” option will do a one-time copy and display of the corresponding DataSet. The “**Update**” button will refresh the shown view of the data, and the “**Auto**” will refresh the data at a preset interval as determined by the slider at the bottom of the live data pane. The minimum display refresh period is on the order of 5-10 seconds. The “**Record**” button will send a copy of the live data DataSet to the ISAW tree for further processing.

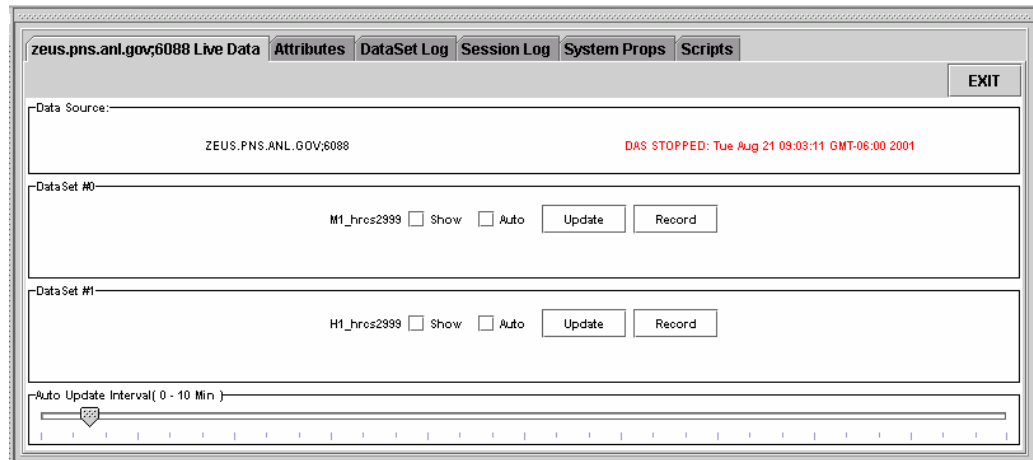
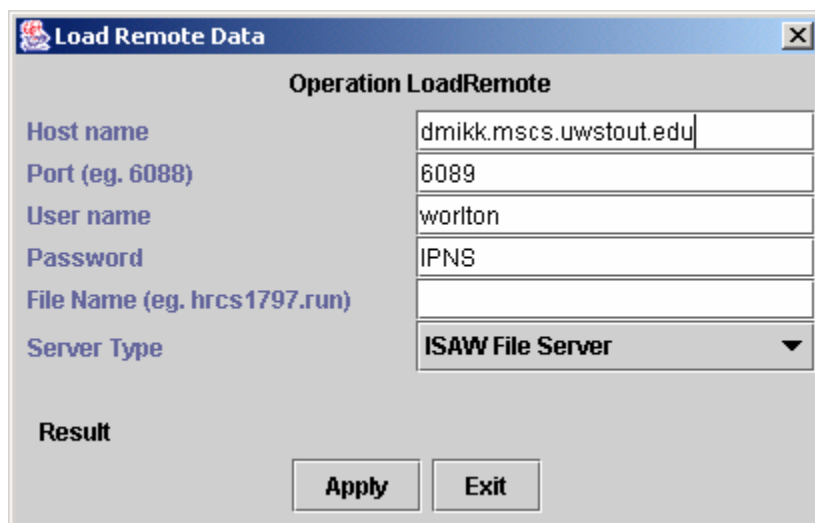


Figure 4- 4. The live data display dialog box. DataSet 0 is a monitor DataSet. Checking the “Show” box on the second DataSet causes ISAW to read the remote data and pop up a view of the DataSet. Checking the “Auto” box would cause ISAW to read the remote data at the interval selected by the slider at the bottom.

### Remote file servers

ISAW can retrieve datasets from remote file servers. The names, addresses, and port numbers of available remote file servers are specified in `IsawProps.dat`. Connection to a remote file server also requires a username and password. When you attempt to connect to one of the remote file servers on the menu, a dialog box will appear asking for all the information needed to make the connection, including username and password.

## READING DATA



The image shows a Windows-style dialog box titled "Load Remote Data". Inside the dialog, there is a section titled "Operation LoadRemote". Below this title, there are several input fields and a dropdown menu. The fields are labeled "Host name", "Port (eg. 6088)", "User name", "Password", and "File Name (eg. hrcs1797.run)". The "Host name" field contains the text "dmikk.mscs.uwstout.edu". The "Port" field contains "6088". The "User name" field contains "worlton". The "Password" field contains "IPNS". The "File Name" field is empty. Below these fields is a dropdown menu labeled "Server Type" which currently shows "ISAW File Server". At the bottom of the dialog, there is a "Result" label and two buttons: "Apply" and "Exit".

Field	Value
Host name	dmikk.mscs.uwstout.edu
Port (eg. 6088)	6088
User name	worlton
Password	IPNS
File Name (eg. hrcs1797.run)	
Server Type	ISAW File Server

### Saving Data

There are currently two ways of saving ISAW data. The first is by saving the ISAW session including all objects. This is possible because all the objects in ISAW are Java serializable objects. The serialized format is not guaranteed to remain the same in future versions of Java, so this may not be a good method for long-term storage of ISAW data. Currently, the only other way of saving data from ISAW is to export it as a GSAS file. The GSAS data format consists of a title card for the file plus a title card and data for each detector bank. The data is written ten integers per row of data. The bank number is used to obtain information from another file that contains calibration information for that bank. To ensure that users know which detector group a bank number refers to, we include comment cards at the beginning of the file. It is also possible to save ISAW DataSets as NeXus files.

To save ISAW DataSets as NeXus files, choose the desired DataSet and select "Save As" from the File menu. In the "Save As" dialog box, select "NeXus File" for "Files of type". Then choose a location and a file name and click on the Save button.

## Data Viewers

*ISAW provides several different ways of viewing the data. For large sets of spectra, the image view is useful. The scrolled graph view and the selected graph view are good ways of looking at individual spectra or small groups of spectra. The 3D viewer provides a picture of the relative positions of the sample and detector elements.*

In the Model View Controller paradigm, the model (or data) and the views of the data are separate. Changing the view of the data does not change the data, but changing the data changes views of the data. The viewers are “observers” of the DataSet. Whenever a DataSet changes, it notifies all observers of that DataSet so they can adjust their view.

### Viewer menus

Each of the different viewers contains a menu bar which contains some functions common to the different viewers, but may also contain some functions which are specific to the current viewer. The viewer *File menu* can be used to close a view and/or save a DataSet modified by viewer operations. The viewer *Edit menu* can be used to sum spectra, delete selected spectra from the DataSet, or change the sort order of the spectra. The *View menu* allows changing the type of view or making temporary axis transforms. The viewer *Options menu* allows you to choose a viewing style. The default value of some of the viewing options can be set through the ISAWprops.dat file. The viewer Options menu may also contain options specific to a particular view.

### The Image Viewer

The image viewer creates an image consisting of a line or block of lines for each detector group. If there are more detector groups than raster lines in the image, scroll bars appear to enable viewing all spectra. The color of each block of lines corresponds to the intensity or number of counts at that time-of-flight (or other independent variable). The data is normally rebinned horizontally to match the number of pixels available. It is also possible to select horizontal scrolling so that each pixel corresponds to one channel or x value.

---

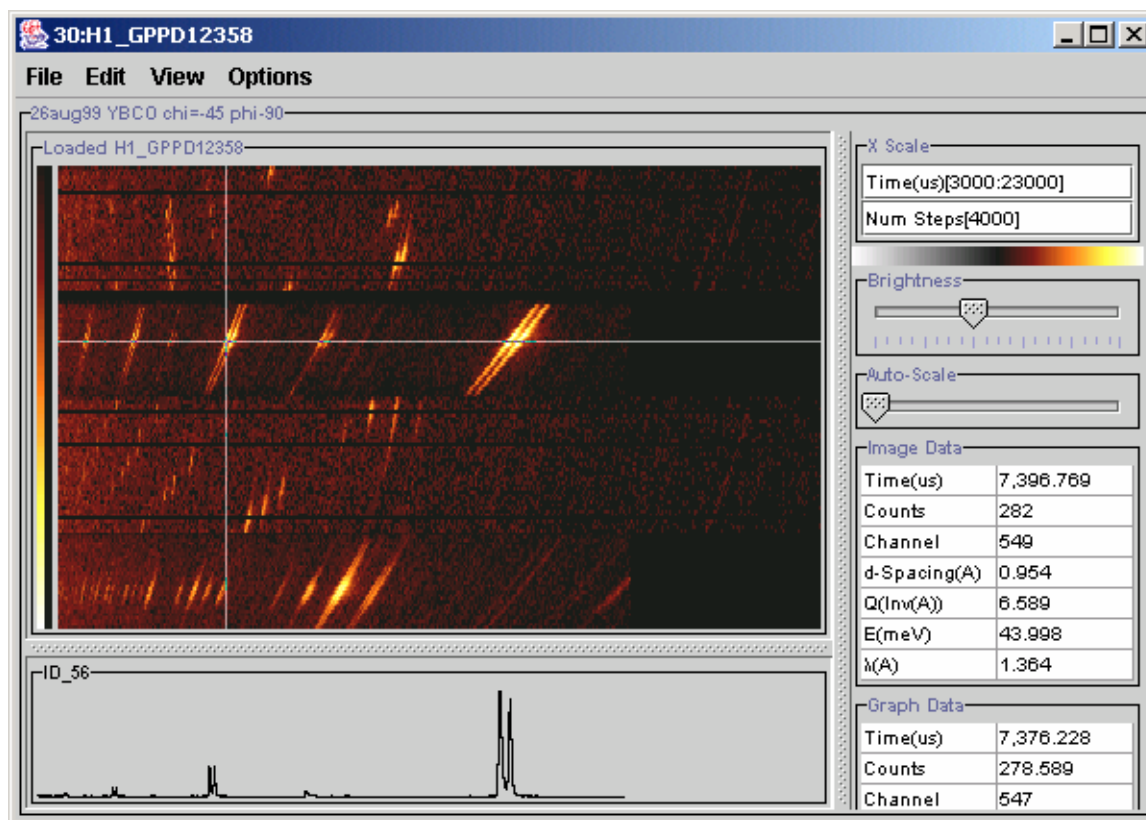


Figure 5- 1. An ISAW image view of time-of-flight neutron scattering data.

### Line graphs

Below the image there is a graph area for displaying line graph(s) of spectra selected from the image. Initially the line graph includes the spectrum that is at the current cursor position, but it is possible to select additional spectra and the graph area will then show more than one line graph. You select the spectrum pointed at by the cursor by typing "s" on the keyboard. To select additional spectra, move the cursor over another line and type "alt-s". To select a block of contiguous spectra, type "Shift-s" after having selected the beginning spectrum of the block using "s" or "alt-s". You can also use the tree view to highlight spectra and then use the right-click menu to mark the highlighted spectra as selected.

### Controls

On the right side of the image view, there is an area for controls and cursor readouts. The horizontal range at the top right is adjusted by clicking in the text area and typing a new range maximum and/or minimum. The second control is for horizontal binning. Binning may be adjusted by clicking on the number of bins and retyping it. The third control is for color intensity. A color scale is shown above the slider used to adjust the brightness. The color scale shows a range of colors for both positive and negative values. If the data has no negative values, only half of the color scale shown will apply.

The "Auto-scale" control at the bottom of the controls region applies to the line graph. The line graph usually is normalized to the highest point, but the scale factor can be adjusted by moving the slider.



**Cursor Readouts**

The image view includes an area for two cursor readouts, one for the image area and one for the line graph area. When the cursor is in the image area, both sets of readouts are controlled by the cursor position. When the cursor is in the graph area, it does not affect the cursor readout for the image area. It is sometimes difficult to select the desired spectrum if there is only one horizontal pixel per detector group. This problem is solved by allowing the user to step through the spectra one at a time using the arrow keys. When the horizontal axis is time-of-flight, the cursor readout also displays a number of equivalent values such as d-value, wavelength, etc. For other choices of x-axis, only the x-axis value and counts at the cursor location are shown.

**Sort order display**

The vertical bar at the left of the image illustrates the “sort order” of the spectra. Spectra are initially sorted according to their group ID, but they may be sorted by any attribute. Use the Edit menu of the viewer to resort the spectra and then observe the difference in the sort order display bar. For more complex sorts involving up to three attributes, you can choose “Sort on group attributes” from the “Edit List” submenu of the Operations menu of the ISAW GUI.

**Selected spectrum marker**

Between the sort order bar and the image area there is an area for marking selected spectra. When spectra are selected, white bars will appear next to the corresponding spectra. You can experiment with this marker by typing “s”, “alt-s”, or “shift-s” while the cursor is on different spectra. After you select a subset of spectra in a DataSet, you can operate on the selected spectra or on the spectra not selected. The selected spectra will also show in a different color on the views of that DataSet.

**Zoom control**

Spectra may have thousands of values and a lot of detail. To easily allow closer examination of these details, you can “zoom in” to a chosen region. The zoom region is chosen by holding down the middle mouse button while dragging the cursor to a new location. Those with a single-button mouse can select the zoom region by using the mouse button together with the shift key. To unzoom, double-click in the image area. These same methods may be used to zoom into a region of the line graph.

**Auto-scale**

By default, with the scale slider at 0, the graph is scaled to the highest point in the graphed data. When moved off from 0, the scale slider specifies the height of the y-axis as a percentage of the largest data value in the whole DataSet.

**The Scrolled Graph Viewer**

The scrolled graph view generates line graphs of all spectra and stacks them one above another. If there are more than a few spectra, scroll bars will have to be used to get to all the spectra. This doesn’t allow viewing many spectra at once, but is useful for quickly viewing a few spectra at a time. Cursor interaction and zooming are supported in this view in the same way as it is supported in the image view.

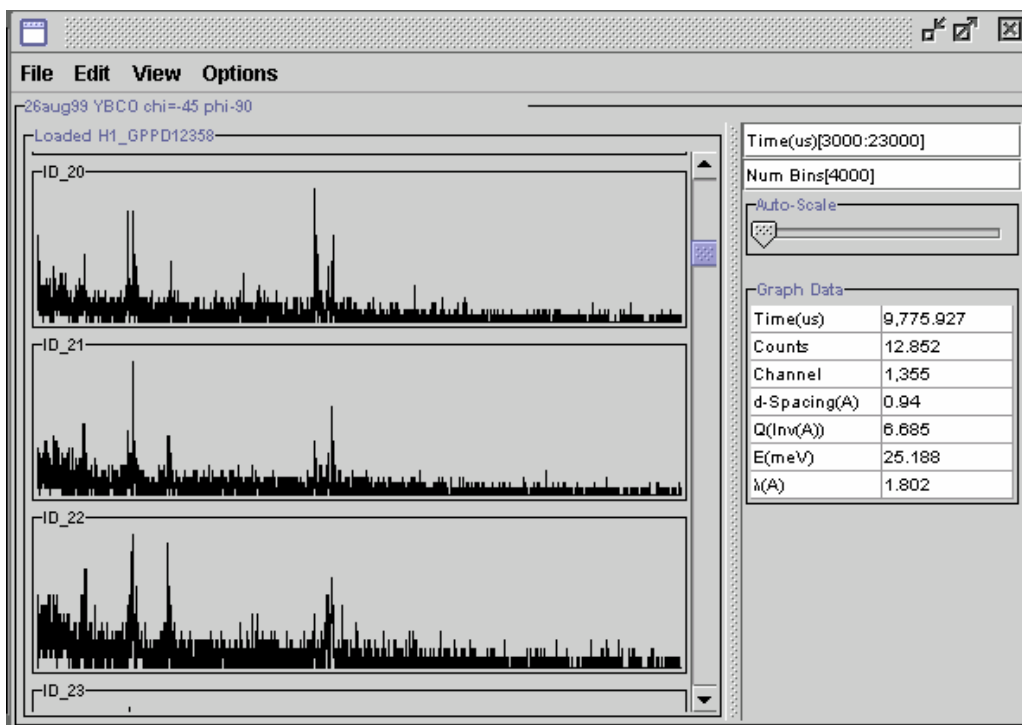


Figure 5- 2. A scrolled graph view of a sample DataSet. The graphs can be scrolled vertically and the cursor can read out the data at any position on any of the graphs.

### The selected graph viewer

The selected graph view allows you to produce a more highly annotated plot of selected spectra. The selected graph view uses the Scientific Graphics Toolkit Software written by Donald W. Denbo (see <http://www.epic.noaa.gov/java/sgt/index.html>). If no spectra have been Selected as discussed above, the Selected Graph View will be empty..

### The 3D Viewer

The 3D viewer provides a three dimensional view of the detector elements and the intensity at that element for a given time-of-flight (or other independent variable). There are controls for changing the viewing position and angle and for scrolling through the data. Colored axes show the beam direction (red) and the other two Cartesian axes.

### The Table Viewer

The table viewer allows you to view, copy, or save tables of data in ASCII format. The table view may be selected from the view menu on the ISAW GUI or on another view of the data.

### Generating a table

- 1) Select "View"->"Table View" from the view menu
- 2) On the table dialog box, Select "x values", "y values", and, if desired, errors from the "Display" column and add these items to the "Display Fields" column.
- 3) Select desired groups.
  - a. Click "Select Group Indices"
  - b. Enter desired group indices (Will not match group numbers. The first group is index 0).
  - c. Click "Apply"
  - d. Click "Exit"
- 4) Check "List Groups Sequentially"
- 5) Check "Make a Table"
- 6) Click on "Display" button.

## Copying data from a table to another application on a PC

- 1) Select desired rows/columns (Use ctrl-a to select the entire table. You can also use the menus to "Select"→"All".)
- 2) Save selected area to the clipboard (You can choose "Copy Set" from the "Select" menu or use ctrl-c to copy selected data to the clipboard)
- 3) Paste the data into the new application (ctrl-v pastes the clipboard data into the new application--Excel, Kaliedagraph, etc.)

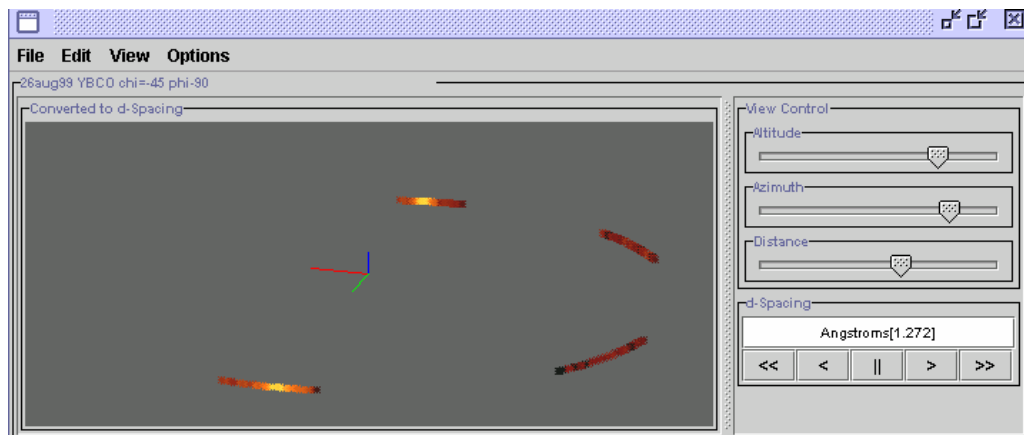


Figure 5- 3. A 3D view of a DataSet. The red line marks the incoming beam direction. The green line marks the axis at 90 degrees to the incident beam in the scattering plane. The blue line indicates the vertical axis. The buttons allow you to either step through the time channels or to scroll steadily through the time channels. The cursor can be used to select a detector element whose attributes will be displayed.

## Command Scripts

### The GUI Elements

#### The Editor Pane

The Editor Pane allows for several lines of instructions to be entered. These instructions can be stored in a file and “OPENed” to be loaded into the Editor Pane. Instructions can also be “SAVED” to a file for later retrieval. The programs can be “RUN” by pressing the **Run Prgm** button.

The text in this pane wraps around.

For the format of the instructions see the following sections.

#### The Immediate Pane

This pane executes one instruction at a time. The instruction that is executed is the line where the **carriage return** was pressed. All the variables from the previous run are still valid so this is like adding an extra instruction to the instructions from the Editor Pane. The structured instructions like for, if –then, on error, etc. are not allowed in the immediate pane.

The text in this pane does not wrap but there are scroll bars.

#### The Buttons

There are five buttons: **Run Prgm**, **Open Prgm**, **Save Prgm**, **Clear** and **Help**. The first three buttons apply to the Editor Pane only. **Run Prgm** runs the program resident in the Editor Pane, **Open Prgm** loads a program into the Editor Pane from a file, and **Save Prgm** saves the text of the Editor Pane to a file. The **Clear** button clears all viewers of Data Sets, the Status Line and the immediate Window. The **Help** button gives some on-line help for this system.

#### The Status Line

The bottom text area is where the error messages and Displayed output appear.

#### Data Types

Variables can only store data of the following types:

Integer  
Float

## COMMAND SCRIPTS

Boolean  
DataSet  
Array - a Java Vector

The first data assigned to a variable determines the data type of the variable. Integer constants have no decimal points while float constants must have a decimal point. The Boolean constants are `true` and `false` (not case sensitive). There are presently no dataset constants.

Array constants are enclosed in square brackets ( `[ ]` ) and elements are separated by commas. Entries may specify a range of integers( ex. `5 : 9` ) and they can be any Java Object data type.

### Other Data types:

ISAW operators can return any Object data type. These return values cannot be stored in a variable but they can be an element of an array or an argument in an argument list.

The dimension of an Array can grow normally by one, but if an array has 5 elements and an instruction assigns a value to the 7<sup>th</sup> element, an error occurs. It is also possible to concatenate ( `&` ) two or more arrays to grow the array.

Variable names are not case sensitive and they follow standard Fortran rules for variables. The leading character must be alphabetic, while the other characters are alphanumeric. The variable names cannot be reserved words like Load, Display, Save, Send, Return, for, end for, if, else, elseif, endif, on, end.

Type checking is incorporated. That is, if a variable stored float data, that variable cannot be assigned an integer data value. Type conversion is NOT done on assignment statements. However, type conversions between numbers are done with the numeric, logical, and relational operations.

### Numeric, Logical, and Relational Operations

- Assignment operator: `=`
- Numeric Operations: `+`, `-`, `*`, `/`, `^` (power)
- Relational Operations: `<`, `<=`, `>`, `>=`, `<>`, `=` or `==`
- String Operations : `&` (Concatenate)
- DataSet Operations : `+`, `-`, `*`, `/` with floats and with other DataSets
- Boolean Operations : And, or, not
- Array Operations: `+`, `-`, `*`, `/`, `&`
  - > `+`, `-`, `*`, `/` with a number applies that operation and number to each element of the Array.
  - > `+`, `-`, `*`, `/` with another Array applies that operation to corresponding elements of an array

- `&`: Concatenates two arrays

These operations obey the standard conventions of precedence of operations.

## Structures

The structures supported by the CommandPane's ScriptProcessor are: for, if, if – else, if – elseif – else, and on error. These structure RESERVED words are not case sensitive.

### For Structure:

The only form of the for statement that is supported is the one shown in the example below:

```
For I in [0,3,5:9]
    Display I
Endfor
```

There are several minor variations that should be noted. The [0,3,5:9] can be an array variable. The values in this array can be strings, float, etc. The variables should all be of the same data type otherwise type mismatch errors occur. Also variables other than I can be used.

### If Structures:

Only the multiline form of these structures is allowed. The “then” is not required at the end of a line starting with the reserved words if or elseif. Nesting with any of the other structures are also supported

#### Examples:

# This is a comment	
If X > 0	if X>=0 and done then
Display X	Display X
Endif	endif
If x< 0	if X>=5 then
Y = x + 1	Y = X + 1
Else	elseif X >=3
Y = x - 1	Y = X
Endif	else
	Y = X - 1
	Endif

### On Error Structures:

The two structures are

```
On Error - End Error
On Error - Else Error - End Error
```

If an error occurs in the On Error block the program will not crash but will transfer control to the block after the Else Error line or if there is no such line, control will be transferred to the statement after the line with the associated End Error instruction. Nesting is supported with this also.

## Intrinsic Operators

The only intrinsic operators are Load, Display, Save, and Send. In the future, Return will also be added. These are reserved words and are not case sensitive.

## Load

There are several forms of the Load instruction:

```
Load ("c:\\Data\\xxx.run")
Load ("c:\\Data\\xxx.run", "Varname")
n = Load("C:\\Data\\xxx.run")
n = Load("c:\\Data\\xxx.run", "Varname")
1 &2 above with parenthesis replaced by spaces
```

All of these create data sets from the file specified. The cases with the second string argument allows the user to specify the variable name for the data sets. In the above cases, the variable names will be Varname [0], Varname [1], .... If the second string argument is not specified, then they will be stored in default variable names. These names are the names that appear in Isaw's data set tree when the file is loaded through Isaw. They look like M1\_xxx, H1\_xxx, etc.

Files besides run files can be loaded with this instruction. Files with extensions "isd", which represents a file storing a data set in the Java Binary format, can be loaded with this instruction. Other files with different extensions will in the future be "loaded" with this instruction.

## Display

There are two forms for the Display command

```
Display expression
Display DataSet_Expression , "SCROLLED_GRAPHHS"
```

Form 1 displays the value of the expression to the Status Line if possible. If the result is a DataSet then a viewer appears to display the data set. Form 2 causes a viewer to appear. The second argument gives the type of viewer that will appear. The second argument must be "IMAGE", "SCROLLED\_GRAPHHS", or "SELECTED\_GRAPHHS"

## Send

There is only one form of the Send command

```
Send DataSetExpression
```

This sends the DataSet to anyone listening for the DataSet. Currently the only listener is Isaw. Isaw will add the resultant DataSet to its data set tree.

## Save

There is only one form of the Save command so far.

```
Save DataSetExpression, filename
```

This saves the result of the DataSetExpression to the given filename. It is saved in Java's Binary format form. See Load above for information on retrieving it.

NOTE: This format for the Data Set may not be stable from one release of Isaw to the next release. Also it may not be stable from one release of Java to the next release.

### Return

Not implemented yet. However, scripts can become operators that may want to return values. Presently, the value returned is the value of the last expression that was evaluated. Most of the time this is null. To return a non-null value without a Return instruction, just place the variable or expression on the last line of the script. For example,

```
X = 3
Y = 2
Display X+Y
#The following will be returned
X-Y
```

### Other Operators

All other commands translated by the CommandPane's ScriptProcessor are packaged as operators. Some operators are "installed" before Isaw or the CommandPane is started. Others can be installed by specifying their location via the "Script\_Path" variable in the IsawProps.dat file. These will be added as either system loads.

These operators can be written in Java and compiled or can be scripts.

### Java-class operators

These operators must subclass GenericOperator or, if the operators are attached to a DataSet, subclass DataSetOperator. The best documentation on these operators is in the javadoc documentation under DataSetTools.operator. The documentation for the getCommand method gives the command name used to invoke this command in a script. One of the constructors gives the order and data types of the arguments and usually tells what the operator does.

### Script Operators

Any script with no errors can become a script operator. The script can be run from the CommandPane by loading the instructions into the CommandPane then pressing the "**Run Prgm**" button.

The command used to invoke this operator as an instruction, if it is installed, in another script is its filename without its path and without its extension. Extra instructions in scripts were added to specify the parameters and prompts for these operators.

```
$ Title = This will appear in dialog boxes and menus
# varname      Data Type      Prompt
$ x              Integer      x=
$ y              String("abcd") y=
```

The Data types that can occur here are any of the supported data types specified above. In addition, several other data types can be used. These were specifically introduced to make user specification of these parameters more user-friendly. These data types are:



## COMMAND SCRIPTS

```
DSFieldString  
InstrumentNameString  
DataDirectoryString  
DSSettableFieldString
```

If there is a parenthesis after the data type specifier, it gives a suggested initial value for this variable when users are allowed to specify the parameters via the dialog box.

Writing Operators:

Examples are worth a thousand words. Look at sample java operators and scripts.

### Writing Java operators

There is a slight difference between DataSetOperators and GenericOperators. In DataSetOperators, the first data set is NOT a parameter so use addDataSet instead of add parameter for that argument. This difference is transparent when writing scripts using these operators.

Furthermore, these operators can implement Iobservable and/or Customizer if the required add methods would be needed. The system automatically adds any available listeners of these types to your operator.

### Input-Output Considerations:

Functions and Subroutines need inputs and produce outputs. The input values must be via parameters. However, getting the results back to the calling program requires some additional comments.

### [Input-]Output Parameters:

Output parameters can only have the data types of DataSet and Array(in Java Vector). Other unsupported data types may also be used for sending results back in java-class operators if the code is written carefully.

### Return value:

The operator can be a function and have some value floating around when the operator is through. Java-class operators are required to do this. See the Return statement for scripts.

### Considerations for other cases:

If an operator has several results that must be sent back to the calling program, these results could be stuffed into an Array( in Java, Vector).

### Interface to Isaw

The CommandPane was specifically designed to be integrated with Isaw. Most menu choices in Isaw translate to commands in the CommandPane.

### Isaw to CommandPane:

All DataSets in Isaw's DataSet tree can be accessed by the scripts in the CommandPane according to the following convention:

If the name on the tree is 25:M1\_hrcs2204 , then the script must refer to it as IsawDS25.

## COMMAND SCRIPTS

Again, as mentioned in the Load instruction, the default name given to the variable storing a data set after the load instruction would be M1\_hrcs2204 because that is how the data set appears in Isaw's data set tree.

### **Isaw to ScriptProcessor:**

There is a submenu option, "Load Scripts", under the Isaw's File menu option. This will execute the script from the filename selected. Unfortunately, none of the Display instructions will appear anywhere presently.

### **CommandPane to Isaw:**

The Send instruction will send any data set to Isaw. Usually the send instruction will always send the data set to Isaw if Isaw is running. The CommandPane can run by itself. In that case, the data set does not get sent anywhere.

## Chopper Analysis Package

Custom ISAW routines have been written for analyzing time-of-flight data obtained from a chopper spectrometer[3]. This ChopAnalysis Package (CAP) is a component within the architecture of ISAW. Multiple large data sets, each containing  $\sim 1000$  spectra, can be examined for removal of noise and contamination. Appropriate sample, background and calibration runs are combined and normalized. Afterward, data from detector groups are binned and converted into various scattering functions in momentum- and energy-transfer ( $|\mathbf{Q}|, E$ ) space. Users can analyze the resulting datasets further with additional methods. By virtue of the object-oriented programming (OOP) methods and graphic user interface (GUI) of ISAW, immediate steps of the evaluation are easily displayed in interactive viewers as images or line plots. Repetitive procedures can be executed through command-line scripts. The results can be saved as formatted tables in text files or pasted into other applications.

### History

The High-Resolution Medium-Energy Chopper Spectrometer (HRMECS) is the first IPNS instrument to employ a new data acquisition system (DAS) and enhanced detector electronics as a part of the upgrade plan of HRMECS.[1]. The protocol for data storage and retrieval via the runfiles within the new DAS, written in Java, permits an efficient entry into the Integrated Spectral Analysis Workbench (ISAW) via a database.[2]. A typical HRMECS runfile consists of, besides the instrumental parameters and attributes of sample environment, about 1000 time-of-flight (TOF) detector spectra, each containing up to 4000 channels. The basic steps for initial data reduction include:

- 1) Examine the detector response over all the spectra and, if necessary, remove bad detector spectra for a series of runfiles including sample, background and calibration runs;
- 2) Calculate the incident energy of each run and check for consistency so that background counts are properly subtracted from the sample runs and the TOF spectra are normalized according to the sample scattering cross section, detector size and relative efficiency; and
- 3) Rebin data histograms in counts over TOF channels and detector positions into appropriate scattering functions in density distribution over the momentum- and energy-transfer ( $|\mathbf{Q}|, E$ ) space.

This procedure is illustrated in Fig. 7-1. The ChopAnalysis Package (CAP) is built within ISAW, which provides all the necessary components for user interface, data control, and data modeling within the Model-View-Controller architecture. Since we expect that this procedure would be repeated for a number of runfile sets that belong to an experiment, ISAW includes a scripting language to control the data processing and the steps are automatically logged. For example, a user may easily recall the script of a procedure, edit a few entries corresponding to a new set of runfiles, and execute a part of or the entire procedure. Export of intermediate DataSets in text or table format is also available.

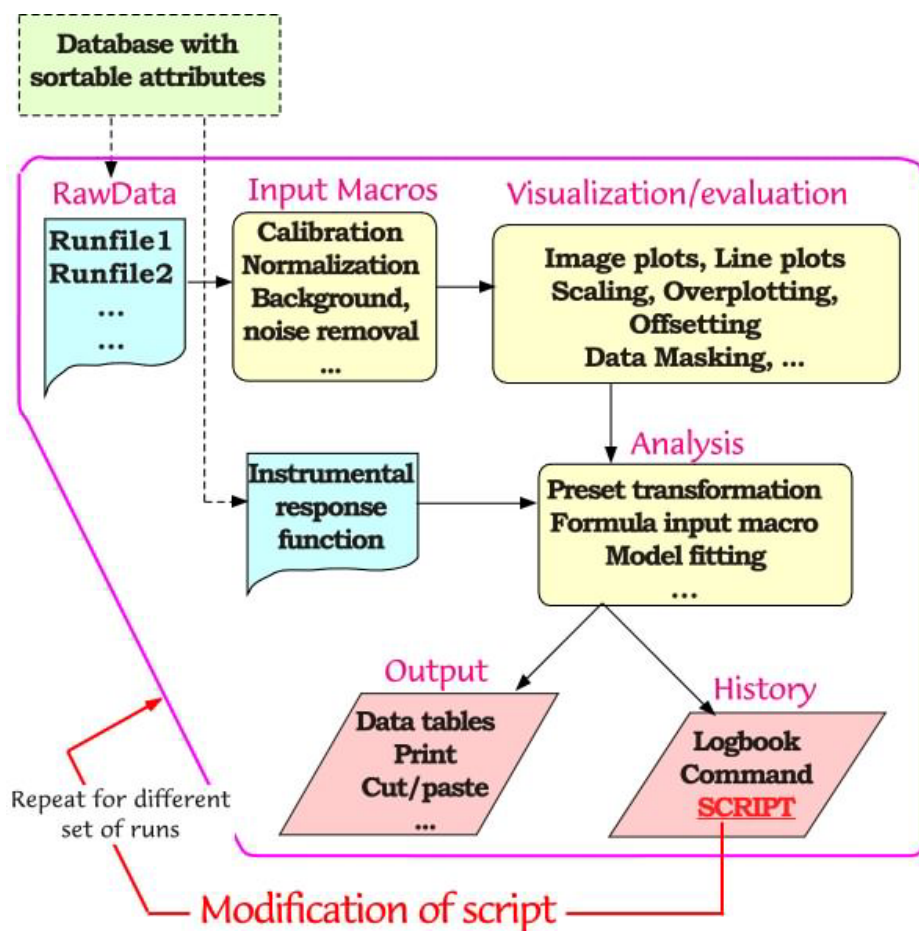


Figure 7-1. The basic data reduction scheme of the ChopAnalysis Package (CAP).

### DataSet Evaluation

Fig. 7-2 displays a typical panel corresponding to detector response in a runfile collected under the pulse-height-spectral mode. The scrollable window on the right is one of many ways to display the detectors' pulse-height spectra (PHS), provided by the IsawGUI.[3]. Furthermore, quantitative analysis of the PHS can be done by invoking the proper model/method belonging to the ISAW internal "Peak class", which is designed to handle various characterization and fitting of peak-like spectra. Then the result is shown in the left panel, permitting further examination or eventually the setting of the detector discrimination levels according to an acceptance criterion. At present, the software for evaluation of the PHS and setup of discrimination levels has not been

completed, but we would like to emphasize ISAW's powerful features which allow the reuse of existing operators and the easy addition of new operators to perform a variety of spectral analyses.

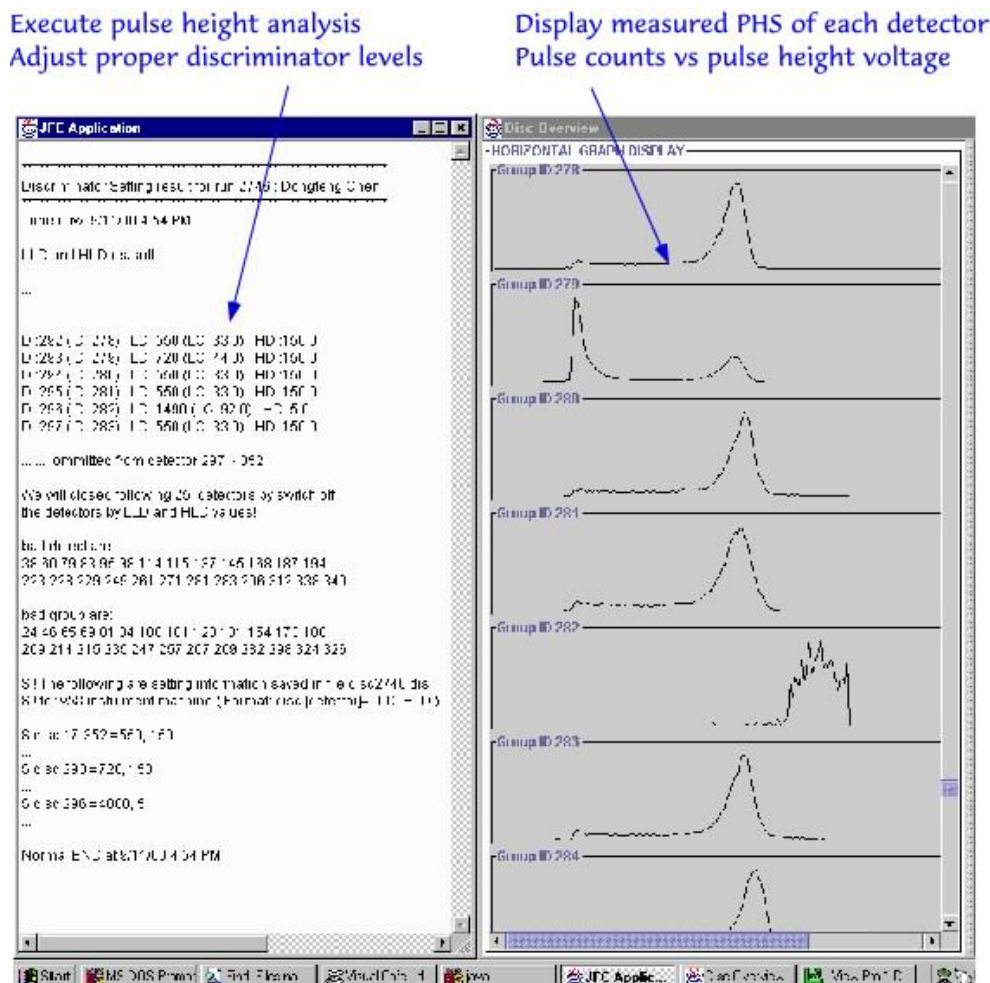


Figure 7-2. A display of detector pulse-height spectra.

Another important capability of ISAW is the efficient display of all of the detector intensity records (~1000 per runfile) using an image plot with the ability to zoom-in a selected group of TOF profiles. Moreover, it provides a side-by-side comparison of the data before and after applying a filter/operation. For example, a macro of "Remove bad detector data" can be executed within the "Operations" menu, and the result is shown in Fig. 3. If the outcome is desirable, a new DataSet with bad detectors removed is created, and the corresponding script for such an operation is also generated for future treatment of other runfiles.

#### Combining Multiple Runs and Conversion to Various Scattering Functions

Fig. 7-4 shows the calibration of the incident energy, normalization of background run(s) against sample run(s), and calibration of the intensities according to a vanadium standard run, all done in a menu-driven single step.

## CHOPPER ANALYSIS PACKAGE

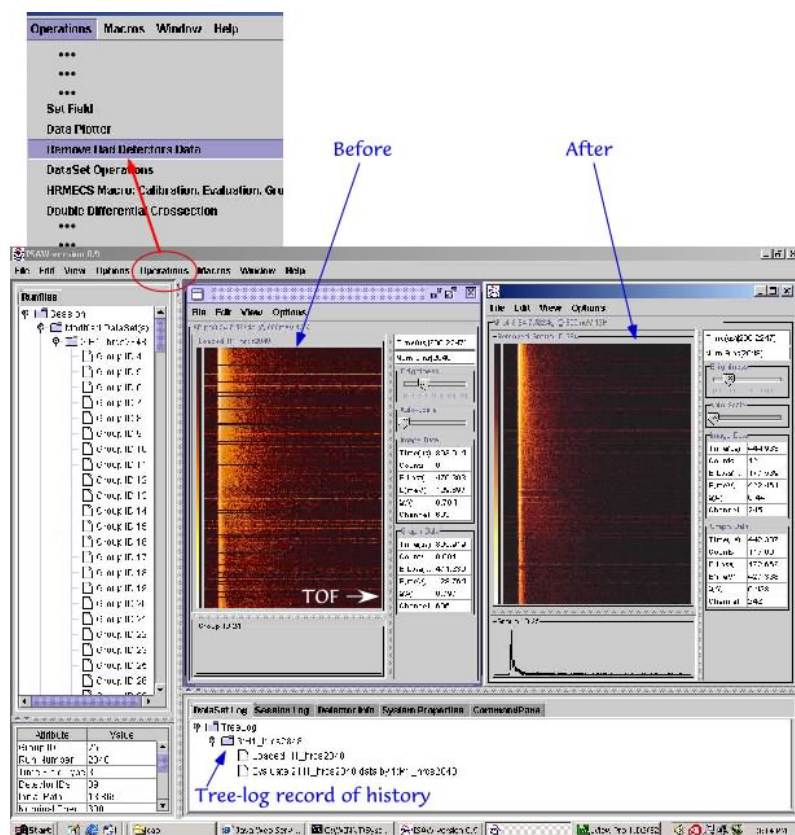


Figure 7-3. Removal of bad detector data.

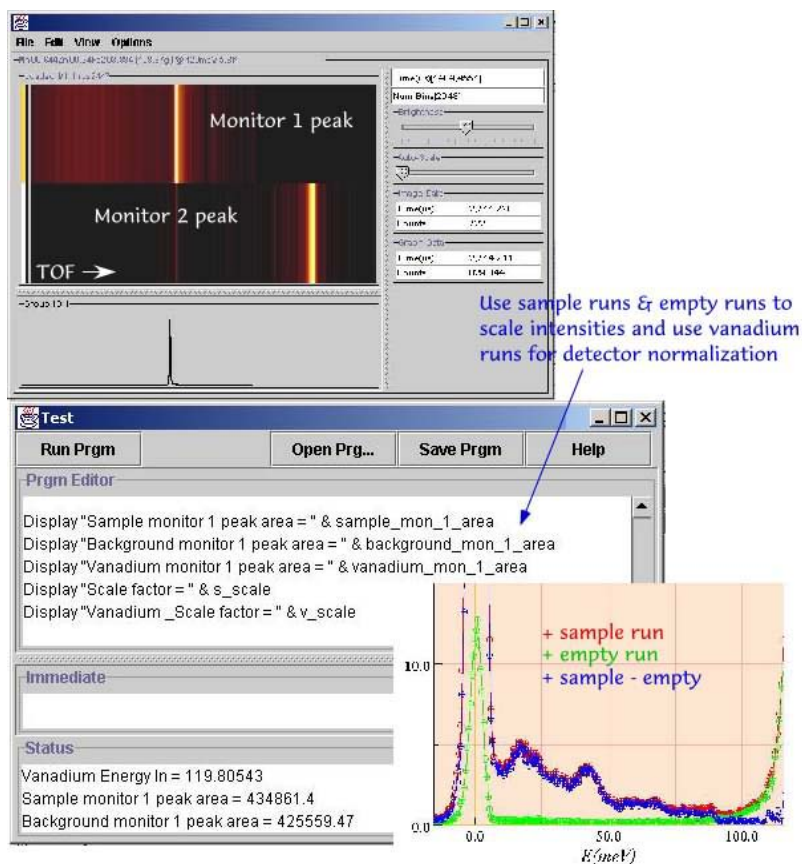


Figure 7-4. Calibrate incident energy and normalize runs.

The rebinning of data from TOF to (Q,E) space is shown in Fig. 7-5 and subsequently converting the data to different kinds of scattering functions is shown in Fig. 7-6.



# CHOPPER ANALYSIS PACKAGE

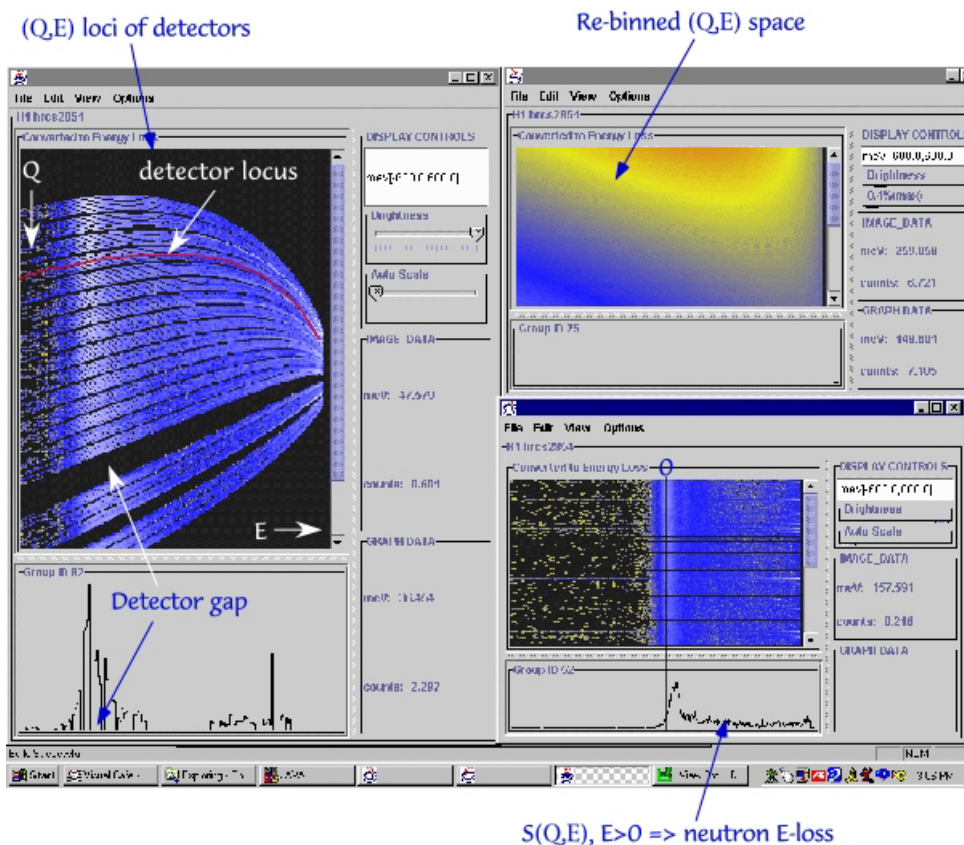


Figure 7-5. Rebinning data from TOF to (Q,E) space.

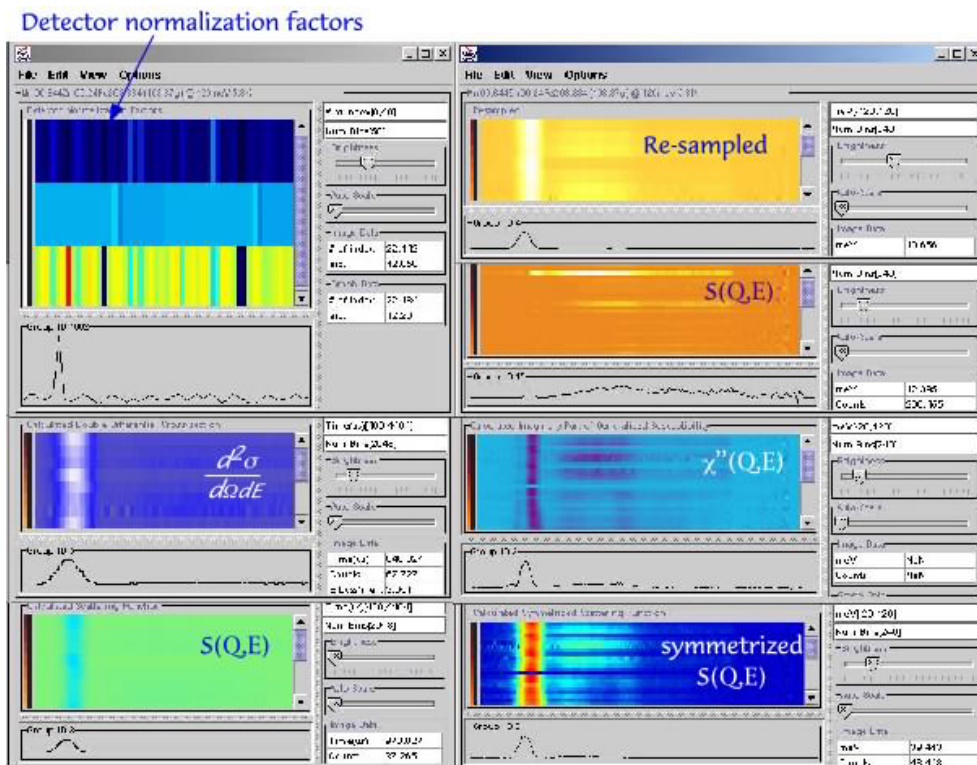


Figure 7-6. Conversion of data into different scattering functions.



**Future Plans**

We have discussed the basic tools of the ChopAnalysis Package (CAP) that permit data visualization and reduction of TOF data of a chopper spectrometer to scattering functions within ISAW. Currently this procedure is being tested and refined. Many additional features are planned for future versions, such as run-time monitoring of detector performance, multiple-scattering and multi-phonon corrections, generation of instrumental resolution, and more extensive curve fitting capabilities. CAP is the first software package developed for chopper spectrometers within ISAW. Similar packages will be introduced for other IPNS instruments in parallel with the upgrade of their DAS. Eventually, relevant data sets collected using different instruments could be combined and incorporated in order to produce maximum information benefiting a scientific study. For example, a cut at the elastic region of a scattering function,  $S(\mathbf{Q}, E=0)$  obtained from a spectrometer would provide a useful comparison with the  $S(\mathbf{Q})$  obtained from a total-scattering diffractometer. Also, scattering functions covering different energy ranges with distinct resolutions obtained from direct- and inverse-geometry spectrometers could be combined for an investigation of evolution of excitations over a wide range of  $(\mathbf{Q}, E)$  space. This is within the design capability of ISAW since the software uses the same data and basic operator classes for all instruments. ISAW can, in principle, be expanded for use by neutron TOF instruments at other facilities because of its platform independence and Internet accessibility.



## Building DataSets

Dennis Mikkelsen

### INTRODUCTION

A DataSet object, as used by ISAW, is a container object that contains zero or more Data objects. Each Data object represents a tabulated function or histogram using a collection of "y" values and corresponding "x" values. Both the containing DataSet and each Data object that it contains also hold several types of auxiliary information. Some of the auxiliary information is in the form of a fixed set of data fields in the objects. Some of it is in an extensible list of "attributes" maintained by each object. Various operations can be performed on a DataSet and the DataSet includes an extensible list of operators that can operate on the DataSet. Finally, the DataSet keeps a "log" of the operations that have been applied to the DataSet.

#### Major steps in building a DataSet:

1. Construct the empty DataSet complete with appropriate operators.
2. Add "attributes" to the DataSet.
3. Construct a Data block to add to the DataSet.
4. Add "attributes" to the Data block.
5. Add the Data block to the DataSet.

These steps would usually be done in the order listed, with the last three being repeated for each Data block that is added to the DataSet.

#### CREATING A DATA SET:

##### Construct the empty DataSet.

This is very easy for a time-of-flight DataSet. There is a DataSetFactory that can be used to build the empty DataSet and add the needed operators to the DataSet. For a time-of-flight DataSet this is used as shown below. The "title" parameter that is passed to the constructor of the DataSetFactory will specify what title will be used for subsequent DataSets produced by the factory.

```
DataSetFactory ds_factory = new DataSetFactory( title );
```

## BUILDING DATASETS

```
DataSet ds = ds_factory.getTofDataSet( instrument_type );
```

The `instrument_type` is an integer code for the type of instrument. This is used by the `DataSetFactory` to determine which operators should be added. The values for the integer codes are defined in the file:

```
.../DataSetTools/instruments/InstrumentType.java
```

and currently include:

```
InstrumentType.TOF_DIFFRACTOMETER
InstrumentType.TOF_SCD
InstrumentType.TOF_SAD
InstrumentType.TOF_DG_SPECTROMETER
InstrumentType.TOF_IDG_SPECTROMETER
InstrumentType.TOF_REFLECTROMETER
```

At this time `DataSetFactory` provides a larger set of operations for the types `TOF_DIFFRACTOMETER` and `TOF_DG_SPECTROMETER`. Support, by way of special operators for the other instrument types is still being developed. In all cases, very basic operations such as add, subtract, multiply and divide by `DataSets` and scalars are included. If you are constructing a "generic" `DataSet` with axis labels and units other than those for a time-of-flight instrument, you can use a different constructor for the `DataSetFactory` such as:

```
DataSetFactory factory = new DataSetFactory( title,
                                             "Angstroms",
                                             "d-Spacing",
                                             "Counts",
                                             "Scattering Intensity" );

DataSet ds = factory.getDataSet();
```

In this case, the factory will produce `DataSets` with the given title, axis units and labels. The method "getDataSet" will only add the generic operators to the `DataSet`, not the operators specific to time-of-flight `DataSets`.

### Add "attributes" to the DataSet.

Attributes can be added to `DataSets` and `Data` blocks at any time. It's probably best to add the attributes you'll need in an organized manner at the time that you are constructing the `DataSet`. Attributes are name, value pairs where the value can be things like an integer, float, array of integers, character string, etc. Attributes are classes that are derived from the abstract base class defined in `.../DataSetTools/dataset/Attribute.java`. This file also contains a list of the names that we have been using for the attributes. The names are given by "constant" strings. Since each attribute is stored in it's own object, it is usually necessary to create the attribute objects as they are added to the `DataSet` (or `Data` block). If a few individual attributes are being added to the `DataSet` (or `Data` block) they can be added using the `setAttribute( attribute )` method. For example, assuming that the original data file name is to be stored as an attribute of the `DataSet`, you could write:

```
ds.setAttribute( new StringAttribute( Attribute.FILE_NAME, file_name ) );
```

to set an attribute for the file name in DataSet "ds". This assumes that the variable `file_name` is a string containing the file name. This will construct a new `StringAttribute` object with the name of the attribute given by the constant `Attribute.FILE_NAME = "File"` and the value of the attribute given by the `file_name` string. Other attributes are treated similarly.

```
ds.setAttribute( new IntAttribute(Attribute.NUMBER_OF_PULSES,
num_pulses));
```

The attribute can also be constructed separately and then set in the DataSet like:

```
int_attr = new IntAttribute( Attribute.NUMBER_OF_PULSES, num_pulses );
ds.setAttribute( int_attr );
```

Finally, there are also routines to get and set the entire list of attributes at once, but the routines to get and set individual attributes are actually more efficient to use in most cases.

### Construct a Data object.

The three most crucial pieces of information held in each Data object are the list of y-values, an XScale object specifying the corresponding x-values and a unique integer ID. These three pieces of information are needed by the constructor for a Data object. The y-values are just an array of type `float[]` and the ID is just an integer value. However, the XScale is an object that either contains the x-values, or contains enough information to calculate uniformly spaced x-values. The x-values are stored in an XScale object for space efficiency. That is, in many cases the x-values associated with a Data block are evenly spaced. In that case, they can be easily calculated as needed based on the first point, the last point and the number of points. Since we may have thousands of spectra with thousands of y-values in each, it would be a serious waste of space to store corresponding evenly spaced x-values in such cases. In this case, the x-values can be stored in a `UniformXScale` object, derived from an XScale object. For example, a uniform XScale object with 101 points evenly spaced on the interval  $[0,10]$  can be constructed as:

```
XScale x_scale = new UniformXScale( 0, 10, 101 );
```

If the x-values are not evenly spaced, a `VariableXScale` object can be used to explicitly store all of the x-values. Specifically, if an array of floats named "x-vals" contained the x-values we could create a `VariableXScale` object as follows:

```
XScale x_scale = new VariableXScale( x-vals );
```

In either case, software using the `x_scale` can get at information such as the min, max, number of points and the actual x-values using the methods of the base class XScale. For a time-of-flight Data object, the operators assume that the times are specified in microseconds. It also should be noted, that Data objects are used to store either histogram data, or tabulated function data. These two cases are distinguished based on the relationship between the number of x-values and the number of y-values. Specifically, for a tabulated function Data object, the number of x-values will be the same as the number of y-values. In this case, the y-values give the value of a function at the corresponding x-value. On the other hand, for a histogram, the Data object records the x-values at the boundaries of the histogram bins. The y-values are considered to be the y-values at the bin centers. Thus for

histogram data, the number of x-values is one more than the number of y-values. The number x-values is restricted by the Data object constructor to be either the number of y-values, or the number of y-values plus one.

An example of building a simple Data block for the function  $y = (x/10)^2$  on the interval  $[0, 49]$ , with ID = 1 is given below:

```
float y_values[] = new float[50];
XScale x_scale = new UniformXScale( 0, 49, 50 );
for ( int i = 0; i < 50; i++ )
    y_values[i] = (i/10.0) * (i/10.0)

Data data = Data.getInstance( x_scale, y_values, 1 );
```

### Add attributes to Data object.

Both DataSet objects and the Data objects that they contain implement the IAttributeList interface. As a result, attributes are added to Data objects in exactly the same way as they are added to DataSets. For example, if data is a Data object, we could add an attribute specifying that the initial energy was 120.0 as follows:

```
data.setAttribute( new FloatAttribute( Attribute.ENERGY_IN, 120.0f ) );
```

### Add the Data object to the DataSet.

Once a Data object has been constructed, and its attributes set, it should be added to a DataSet. For example, to add a Data object "data" to a DataSet "ds" just do:

```
ds.addData_entry( data );
```

### ATTRIBUTES NEEDED IN A DATA SET and DATA object:

Although the above discussion describes how to construct a Data block and DataSet, more information is needed to construct a DataSet to hold time-of-flight data in a way that will allow useful operations to be done on the Data. In particular, most of the "interesting" operators for neutron scattering rely on specific attributes of the DataSet and Data objects. The attributes that are currently used by various operators include:

Attribute.DETECTOR_POS	<- object
Attribute.INITIAL_PATH	<- float
Attribute.ENERGY_IN	<- float
Attribute.NUMBER_OF_PULSES	<- int
Attribute.SOLID_ANGLE	<- float
Attribute.DELTA_2THETA	<- float
Attribute.RAW_ANGLE	<- float

To allow for comparing and scaling DataSets, some measure of the number of neutrons that hit the sample is needed. For use in scripts, this should probably be the number of pulses, at least that is what has been used for GPPD. If the number of pulses is not directly available, it could possibly be

approximated based on a start time and end time. At any rate, it would be useful to have the number of pulses stored as a DataSet attribute for any instrument.

The attributes listed above are primarily used as attributes of each Data object. The attributes are listed in decreasing order of importance. Interpreting the time-of-flight data almost always requires the effective detector position. The convention used in the DataSetTools package is that the effective detector position gives the position of the detector from the center of the sample. Since there are different ways of specifying this position, a class was constructed to hold the position information and provide some extra information as needed. A "Position3D" object contains a 3D position, specified in any of the usual coordinate systems, Cartesian, cylindrical or spherical. There are methods to get and set the position in any of these coordinate systems, as well as some additional convenience routines.

The convention for the instruments at IPNS is that the coordinate system has its origin at the sample position, the x-axis points in the direction the incident beam is traveling, the y-axis is horizontal, perpendicular to the incident beam and z-axis is perpendicular to the earth's surface. This is also the convention followed by the DataSetTools package. Unfortunately, that coordinate system is somewhat inconvenient for describing the scattering angle (the angle between the positive x-axis and the vector from the sample to the detector). Since the operators frequently need to use the scattering angle a class "DetectorPosition" was derived from the Position3D class. The DetectorPosition class adds a method to get the scattering angle, and so it should be used to represent the position of the detector relative to the sample. An example of code to set a detector position attribute corresponding to a detector that is at an angle of 50 degrees, 0.1 meter above the xy plane, and above a horizontal circle of radius 4.0 meters centered at the sample is shown below:

```
DetectorPosition position = new DetectorPosition();
float angle      = 50.0f * (float)(Math.PI / 180.0);
float final_path = 4.0f;
float height     = 0.1f;
position.setCylindricalCoords( final_path, angle, height );
data.setAttribute( new DetPosAttribute( Attribute.DETECTOR_POS,
                                     position ) );
```

Lengths are assumed to be in meters, and angles are stored in radians. If the detector position is easier to specify in Cartesian coordinates, (x, y, z), the method position.setCartesianCoords( x, y, z) can be used instead.

The initial path attribute is needed for the diffractometer instruments. The initial flight path is the source to sample distance in meters. If this can be obtained, as say the float variable "length", it is easily added to the Data block as:

```
data.setAttribute( new FloatAttribute( Attribute.INITIAL_PATH, length
) );
```

The operators to process data from direct geometry spectrometers require the initial energy of the neutrons incident on the sample. The initial energy is assumed to be in meV. It is often necessary to calibrate this value, but at least some initial approximation will be needed by these operators. The more advanced operators for direct geometry spectrometers will require the number of pulses to be

stored with each Data block, in addition to being stored with the DataSet as a whole. Finally, these operators need the solid angle subtended by the detector group, measured in steradians.

The operator to produce a display of  $S(Q,E)$  for spectrometers will need an approximate value for the interval of scattering angles covered by each detector. That is, each detector has non-zero dimensions. Consequently, even though the detector might be nominally at say 50 degrees, it actually covers some interval, say 49.95 to 50.05 degrees. Some approximation to the range of angles covered should be stored in a DELTA\_2THETA attribute. This value is assumed to be stored in degrees.

The operator to produce a "TrueAngle" display of a DataSet requires the DELTA\_2THETA attribute, as well as the RAW\_ANGLE. The RAW\_ANGLE is the actual physical scattering angle for the detector, without regard to time-focusing. ( Time focusing may adjust the raw angle to a different effective angle. ) A DETECTOR\_POS attribute is assumed to hold the effective 3D position of the detector, while a RAW\_ANGLE attribute is assumed to hold the physical, unfocused scattering angle.

#### **A DATA RETRIEVER:**

In order to easily work with different sources of data, such as IPNS runfiles, Nexus files, data acquisition hardware, etc. the system was designed to access data through subclasses of the abstract class `.../DataSetTools/retriever/Retriever.java`. Currently the only derived class is a `RunfileRetriever` that accesses IPNS runfiles. New data sources should be supported by making a new class derived from the `Retriever` class, since in that way, all data sources can be used in the same way. The `Retriever` class is quite simple:

The constructor accepts a string giving the name of the data source. For a data file, this would most likely be the file name, and the file would most likely be opened in the constructor. The `Retriever` class then provides three methods, a method to get the number of DataSets available from the source, a method to get the type of each available DataSet ( `MONITOR_DATA_SET` or `HISTOGRAM_DATA_SET` ) and a method to get a specific DataSet from the source. For the special case of the IPNS runfile retriever this gets used as simply as:

```
RunfileRetriever rr    = new RunfileRetriever( "gppd9898.run" );
DataSet A_monitor_ds   = rr.getDataSet( 0 );
DataSet A_histogram_ds = rr.getDataSet( 1 );
```

where we've used the simplifying assumption that the "zeroth" DataSet is always the monitor DataSet and the "first" always the first histogram DataSet. These simplifying assumptions make it unnecessary to find out the number of DataSets and find out their types before reading.

As other types of files or data sources are supported, the `Retriever` class may need to expand slightly. However it is best to keep this class as simple as possible, since any new functionality introduced in the `Retriever` will have to be supported by ALL types of retrievers.

#### **AN EXAMPLE:**

A simple program to demonstrate building a DataSet is in the file:

```
.../DataSetTools/trial/BuildDataSetDemo.java
```

## BUILDING DATASETS

in the latest version of DataSetTools. It can be compiled from within the directory containing it using:

```
javac BuildDataSetDemo.java
```

and then can be run using

```
java BuildDataSetDemo
```

Assuming that all PATH and CLASSPATH values have been set properly. The code for the demo is listed below:

```
/*
 *  @(#) BuildDataSetDemo.java      1.0   2000/9/19      Dennis Mikkelson
 *
 */
import DataSetTools.dataset.*;
import DataSetTools.viewer.*;
import DataSetTools.math.*;

/**
 * This class provides a basic demo of how to construct a DataSet.
 */
public class BuildDataSetDemo
{
    /**
     * This method builds a simple DataSet with a collection of 10 sine
     waves.
     *
     * @return A sample DataSet with 10 sine waves.
     */
    public DataSet BuildDataSet()
    {
        //
        // 1. Use a "factory" to construct a DataSet with operators -----
        //
        DataSetFactory factory = new DataSetFactory( "Collection of Sine
Waves",
                                                    "time",
                                                    "milli-seconds",
                                                    "signal level",
                                                    "volts" );

        DataSet new_ds = factory.getDataSet();

        //
        // 2. Add attributes, as needed to the DataSet -----
        //
        new_ds.setAttribute( new StringAttribute( Attribute.FILE_NAME,
                                                    "BuildDataSetDemo.java" ) );
        new_ds.setAttribute( new IntAttribute( Attribute.NUMBER_OF_PULSES,
10000 ) );

        //
        // Now, repeatedly construct and add Data blocks to the DataSet
    }
}
```



## BUILDING DATASETS

```
//
Data      data;    // data block that will hold info on one signal
float[]    y_values; // array to hold the y-values for that signal
XScale     x_scale; // "time channels" for the signal

for ( int id = 1; id < 10; id++ )          // for each id
{
    //
    // 3. Construct a Data object
    //
    x_scale = new UniformXScale( 1, 5, 50 ); // build list of time
channels

    y_values = new float[50];                // build list of counts
    for ( int channel = 0; channel < 50; channel++ )
        y_values[ channel ] = 100*(float)Math.sin( id * channel / 10.0 );

    data = new Data( x_scale, y_values, id );

    //
    // 4. Add attributes as needed to the Data block
    //
                                // "simple" energy in attribute
    data.setAttribute( new FloatAttribute( Attribute.ENERGY_IN, 120.0f )
);

                                // more complicated, position
                                // attribute has a position
                                // object as it's value
    DetectorPosition position = new DetectorPosition();
    float angle      = 50.0f * (float)(Math.PI / 180.0);
    float final_path = 4.0f;
    float height     = 0.1f;
    position.setCylindricalCoords( final_path, angle, height );
    data.setAttribute( new DetPosAttribute( Attribute.DETECTOR_POS,
                                position ) );

    //
    // 5. Add the Data object to the DataSet
    //
    new_ds.addData_entry( data );
}

return new_ds;
}

/* -----
- */
/**
 * The main program method for this object
 */
public static void main(String args[])
{
    BuildDataSetDemo demo_prog = new BuildDataSetDemo();// create the
class

    DataSet test_ds = demo_prog.BuildDataSet();          // call the method to
```

## BUILDING DATASETS

```

// construct a DataSet
// create a viewer for
// the DataSet
    ViewManager view_manager = new ViewManager( test_ds,
IViewManager.IMAGE );
    }
}
```

## References

1. T. Worlton, A. Chatterjee, D. Mikkelsen, R. Mikkelsen, D. Chen, and C.-K. Loong, Proceedings of the Fifteenth Meeting of the International Collaboration on Advanced Neutron Sources, KEK Proceedings 2000-22, Tsukuba, Japan, 2000) 657-664..
2. A. Chatterjee, T. Worlton, J. Hammonds, D. Mikkelsen, R. Mikkelsen, D. Chen, and C.-K. Loong, *ibid*, 678-684.
3. D. Chen, C.-K. Loong, D. Mikkelsen, R. Mikkelsen, A. Chatterjee, and T. Worlton, *ibid*, 665-670.
4. C.-K. Loong, J. P. Hammonds, D. Yocum, J. C. Nipko, K. Takeuchi, D. Chen, L. I. Donley, G. E. Ostrowski, R. K. Crawford, H. Belch, and R. Kleb, in The Fourteenth Meeting of the International Collaboration on Advanced Neutron Sources, ICANS-XI, edited by J. M. Carpenter (Argonne National Laboratory, 1998 Starved Rock Lodge, Utica, Illinois, 1998), p. 829.

## Index

- array demo.iss, 4
- class files, 6
- Clear, 23
- Command
  - Array Operations, 24
  - Assignment operator, 24
  - Boolean Operations, 24
  - Data Types, 23
  - DataSet Operations, 24
  - Display, 26
  - For, 25
  - If, 25
  - Intrinsic Operators, 26
  - Load, 26
  - Numeric Operations, 24
  - On Error, 25
  - Relational Operations, 24
  - Save, 26
  - Send, 26
  - Status Line, 23
  - String Operations, 24
  - Structures, 25
- command pane, 2
- Data object, 2, 11, 37
- DataDirectoryString, 3, 4
- DataSet object, 2, 37
- DataSet organization, 11
- DataSetFactory object, 11
- Default\_Instrument, 4
- Edit menu, 1
- Editor Pane, 23
- file dialog box, 13
- File menu, 1
- GUI, 1, 2, 3
- HDF, 5, 6, 7
- Help, 23
- Help menu, 2
- InstrumentNameString, 3, 4
- IsawProps.dat, 3, 4, 5, 6, 8, 9
  - default local file locations, 8
  - Live Data Server connections, 8
  - Specifying Remote Data Server connections, 9
  - Viewer Defaults, 9
- ISAWprops.dat, 18
- Java language, 5
  - byte code, 5
  - file types, 5
  - jar file, 5, 6
  - source code, 5
- Macro menu, 2
- MoreSampleData, 4
- NeXus, 5, 6, 7
- observers, 18
- Open Prgm, 23
- Operations menu, 2
- operator, 12
- Options menu, 18
- right click, 11
- Run Prgm, 23
- sample scripts, 6
- Save Prgm, 23
- TCP protocol, 15
- UDP protocol, 15
- View menu, 1
- viewing live data, 2